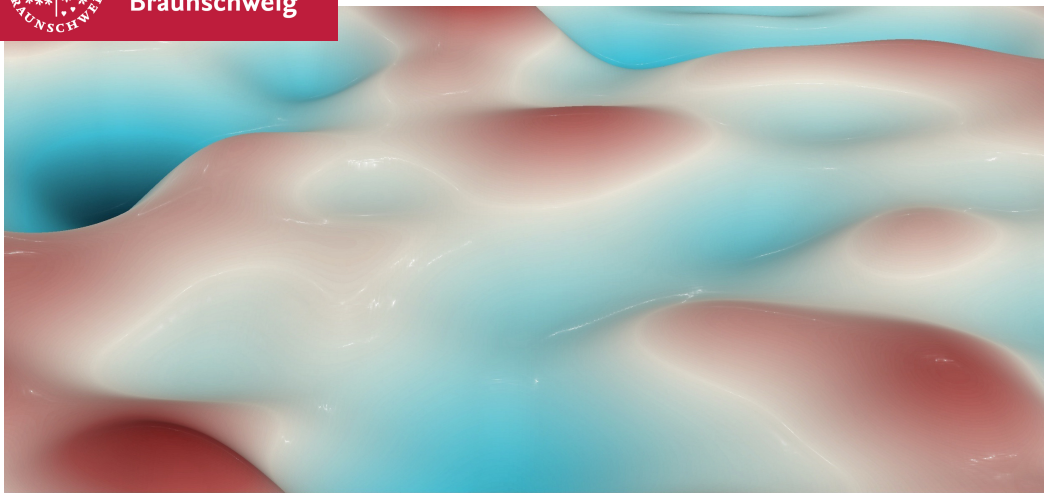
Technische Universität Braunschweig

Institute of Scientific Computing



A Generic Component-Based Software Architecture for the Simulation of Probabilistic Models

Dissertation

Martin Krosche

2013

A Generic Component-Based Software Architecture for the Simulation of Probabilistic Models

Von der
Carl-Friedrich-Gauß-Fakultät
der Technischen Universität Carolo-Wilhelmina zu Braunschweig

zur Erlangung des Grades eines
Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigte Dissertation

von
Martin Krosche
geboren am 11. Juni 1977
in Wolfenbüttel

Eingereicht am:	19. Juli 2012
Disputation am:	13. November 2012
1. Referent:	Prof. Hermann G. Matthies, Ph.D.
2. Referent:	Prof. Habib N. Najm, Ph.D.

2013

The cover shows an optically processed sample of Young's modulus in x-direction which corresponds to an interlayer of a probabilistic model for the material of a three-dimensional laminated composite structure and is defined on the two-dimensional reference plane of that interlayer. For more details, see Section 4.2.1.3 and especially Figure 4.25(a) which presents a further sample.

Abstract

An uncertain behaviour is in the nature of many physical phenomena. This uncertainty has to be quantified for a meaningful prediction by a computer-aided simulation. A stochastic description of the uncertainty carries a physical phenomenon over to a probabilistic model, which is usually solved by numerical schemes.

The present thesis discusses and develops models for challenging uncertain physical phenomena, efficient numerical schemes for a quantification of uncertainties (UQ), and a sustainable and efficient software implementation.

Probabilistic models are often described by stochastic partial differential equations (SPDEs). The stochastic Galerkin method represents the solution of an SPDE by a set of stochastic basis polynomials. A problem-independent choice of basis polynomials typically limits the application to relatively small maximum polynomial degrees. Moreover, many coefficients have to be computed and stored. In this thesis new error-controlled low-rank schemes are presented, which in addition select relevant basis polynomials. In this manner the previously mentioned problems are addressed.

The complexity of a UQ is as well reflected in the software implementation. A sustainable implementation relies on a reuse of software. Here, a software architecture for the simulation of probabilistic models is presented, which is based on distributed generic components. Many of these components are reused in different frameworks (and may also be used beyond a UQ). They can be instantiated in a distributed system many times and are interchangeable at runtime, where the generic aspect is preserved.

Probabilistic models are derived and simulated in this thesis, which for instance describe uncertainties for a composite material and an aircraft design. Among other things, several hundred stochastic dimensions or a long runtime for simulations arise.

Zusammenfassung

Ein unsicheres Verhalten liegt in der Natur vieler physikalischer Phänomene. Diese Unsicherheit muss für eine sinnvolle Prognose durch eine Computer-gestützte Simulation quantifiziert werden. Eine stochastische Beschreibung der Unsicherheit überführt ein physikalisches Phänomen in ein probabilistisches Modell, das üblicherweise durch numerische Verfahren gelöst wird.

Die vorliegende Arbeit behandelt und entwickelt Modelle für anspruchsvolle und mit Unsicherheit behaftete physikalische Phänomene, effiziente numerische Verfahren für eine Unsicherheitsquantifizierung (UQ) und eine nachhaltige und leistungsfähige Software-Umsetzung.

Probabilistische Modelle werden häufig durch stochastische partielle Differentialgleichungen (SPDGLn) beschrieben. Die stochastische Galerkin Methode stellt die Lösung einer SPDGL durch eine endliche Menge an stochastischen Basispolynomen dar. Eine problemunabhängige Wahl von Basispolynomen beschränkt die Anwendung typischerweise auf relativ kleine maximale Polynomgrade. Des Weiteren müssen viele Koeffizienten berechnet und gespeichert werden. In dieser Arbeit werden neue fehlergesteuerte Niedrig-Rang Verfahren vorgestellt, die zudem relevante Basispolynome selektieren. Auf diese Weise wird den zuvor beschriebenen Problemen entgegen gegangen.

Die Komplexität einer UQ schlägt sich ebenso auf die Software-Umsetzung nieder. Eine nachhaltige Umsetzung setzt auf die Wiederverwendbarkeit von Software. Hier wird eine auf verteilten und generischen Komponenten basierende Software-Architektur zur Simulation probabilistischer Modelle vorgestellt. Viele dieser Komponenten werden in verschiedenen Frameworks wiederverwendet (und mögen auch außerhalb einer UQ zum Einsatz kommen). Sie können mehrfach in einem verteilten System instanziiert und zur Laufzeit ausgetauscht werden, wobei der generische Aspekt erhalten bleibt.

Probabilistische Modelle beispielsweise zur Beschreibung von Unsicherheiten in einem Kompositwerkstoff und einem Flugzeugentwurf werden in dieser Arbeit hergeleitet und simuliert. Dabei treten mitunter mehrere hundert stochastische Dimensionen oder lange Simulationslaufzeiten auf.

Acknowledgement

This thesis was developed during my employment as a research associate at the Institute of Scientific Computing of the Technische Universität Braunschweig.

Foremost, I would like to thank Professor Dr. Hermann G. Matthies for the possibility to obtain a doctorate, for the numerous helpful notes and remarks, for free space to independently work on my research, all in all for his support for years.

Further, I would like to thank Dr.-Ing. Rainer Niekamp for all the rich discussions in whatever field of research, for being available whenever it was required, and for the uncomplicated collaboration during all the time at the institute.

I would like to thank Dr.-Ing. Wolfgang Heinze from the Institute of Aircraft Design and Lightweight Structures of the TU Braunschweig for his friendly, helpful, and reliable manner, which was an important factor for a successful cooperation in the CRC 880 (collaborative research center).

Furthermore, I would like to thank my French colleagues Dr.-Ing. Martin Hautefeuille and Dr.-Ing. Christophe Kassiotis — with whom I occasionally shared the office — for the friendly and helpful discussions and the support on the software coFeap.

I kindly thank Professor Dr.-Ing. Ina Schaefer and Professor Habib N. Najm, Ph.D., for being members of the examination board, and for valuable discussions.

I would like to thank M.A. Cosima Meyer for her helpful nature and many detailed annotations in dealing with the English language. Also, I would like to thank my colleagues Dipl.-Inform. Dominik Jürgens (for the maintenance of the institute's computer cluster, too), Dr. rer. nat. Alexander Litvinenko, Dr.-Ing. Anna Kučerová, Dr. rer. nat. Dishi Liu, Dr. rer. nat. Alicia Jürgens-Ortega, Dr. rer. nat. Oliver Pajonk, Dr. rer. nat. Joachim Rang, Dipl.-Ing. Bojana Rosić, Dr.-Ing. Jan Sýkora, and Dr. rer. nat. Elmar Zander, as well as Jun.-Prof. Dr.-Ing. Wolfgang Nowak (for his input on groundwater flow problems) from the Institute for Modelling Hydraulic and Environmental Systems of Universität Stuttgart.

In addition, I would like to thank the CRC 880 (SFB 880) and the project “More Affordable Aircraft through eXtended, Integrated and Mature nUmerical Sizing” (MAAXIMUS) for financial support and the graduate school “Graduiertenkolleg” within the CRC 880 — of which I am a member — for the interdisciplinary exchange.

I would like to thank my family and my best friends for the big support and particularly for understanding that I did not have much time for them.

Finally and separately from the previous passage, I would especially like to thank my wife Laura Teresa Ortega Camarena for her patience and for her beaming smile also in times with little time. Without that, this thesis would not have been possible.

Braunschweig, June 2013 Martin Krosche

Contents

Chapter 1

Introduction

The field of scientific computing is concerned with a computer-aided simulation of physical phenomena to obtain useful predictions of their behaviours. A mathematical model to describe the meaningful characteristics of an appropriate phenomenon is necessary to that end. It may formulate the relation between the actions on a physical system and its reactions [128]. The reactions are usually the unknowns — the solution —, for which the model needs to be solved to enable a prediction.

Plenty of physical phenomena obviously exhibit uncertain characteristics or in other words uncertain parameters, for which the uncertainty has to be modelled for a computer-aided prediction. In contrast, a mathematical model of a physical phenomenon may be uncertain through a lack of knowledge. In both cases, a stochastic formulation of the uncertain parameters leads to a so-called *probabilistic model*.

Probabilistic models and their simulations reveal a number of issues which need to be addressed. A mathematical description for the probabilistic model needs to be established first. This indicates to construct models for the associated uncertain parameters, which may be obtained in a discrete manner approximated by numerical schemes. When the probabilistic model is established it needs to be simulated. The simulation may as well simulate or simply use the models for the uncertain parameters. Howsoever, numerical schemes are also usually involved there. Needless to say the mathematical representations and numerical schemes applied for the modelling and the simulation need to be implemented for a computer-aided handling.

Usually, the simulation of a probabilistic model is highly complex and asks for large computational power and memory supply. (This may be already the case for the modelling of the uncertain parameters.) Efficient mathematical representations and numerical schemes are developed and reflected in efficient data representations and

algorithms. The latter may possess parallel or distributed features to exploit the available hardware resources. Codes relevant for an efficient computation are usually implemented in programming languages with high performance support. A software system to simulate probabilistic models needs to handle the mentioned complexity in a preferably sustainable manner.

The next sections provide brief descriptions of the fields of interest and the corresponding contributions of this thesis. A subsection at the end of each section summarises the particular contributions. Numerical schemes to simulate probabilistic models are outlined in Section 1.1. The realisation of sustainable software with an emphasis on the simulation of probabilistic models is addressed in Section 1.2. A couple of challenging probabilistic models are introduced in Sections 1.3 and 1.4.

1.1 Numerical Schemes Based on Probabilistic Models

Uncertain and location-dependent characteristics of a probabilistic model are represented by random fields [5, 6, 206]. In practice, these fields are mainly discretised by a truncated polynomial chaos expansion (PCE) [217, 73, 128, 129, 146] or a truncated Karhunen-Loève expansion (KLE) [125, 73, 128, 129, 98].

For a quantification of uncertainties statistics of the solution may be directly computed, or a surrogate model for the solution may be determined, from which then statistics can be computed. When statistics are directly formulated as integrals — frequently and in this thesis of high dimension —, numerical integration schemes can be applied. These schemes are based on a sampling of the integrand. In this context they may be summarised under the term of direct integration schemes [128, 99]. Monte Carlo (MC) methods [37, 124, 126, 172] are robust examples for such schemes, their convergence is independent of the dimension but known to be slow. Faster convergence may be obtained by the following schemes, at least up to a higher dimension. While MC methods are based on random sampling, Quasi-Monte Carlo (QMC) methods [37, 172, 123] are based on number theoretic point sequences. In contrast, the Smolyak algorithm [71, 70, 169, 170, 52, 99] constructs a sparse grid of samplepoints from a number of one-dimensional quadrature rules.

A surrogate model is usually obtained through a projection of the solution onto a finite set of stochastic basis functions, which are here stochastic polynomials. Such

a projection may be formulated by a number of integrals [145, 123, 122], which may be approximated by the previously mentioned integration schemes. The stochastic collocation schemes [11, 153, 66, 216, 20, 24, 26, 208, 62] do not follow an integral description but also determine the surrogate model on the basis of samples. The stochastic Galerkin method (SGM) [73, 129, 128, 12, 217, 127, 3, 57, 130] projects a weak formulation of the problem onto a solution space.

Schemes which search for a sparse set of stochastic basis functions or a subspace of low rank to describe the solution were recently developed. These approaches are embedded in projection schemes, stochastic collocation schemes, or in the SGM. A sparse set of stochastic basis functions is focused in the context of stochastic collocation schemes [26, 25, 24, 153, 154, 66, 56], and the SGM [54, 201, 23, 111, 143]. Low-rank approaches are published in the context of projection schemes [122], and the SGM [158, 130, 111, 143, 103]. Another publication [55] may be associated to stochastic collocation schemes.

However, combined approaches in which a low-rank representation for the solution in a sparse set of stochastic basis functions is obtained are rarely addressed [111, 143].

The content of this section is extensively discussed in Chapter 2.

1.1.1 Contributions of this Thesis

The approaches in [111] are a contribution of this thesis. A scheme with successive rank-one updates is derived for an SGM-discretised stochastic linear elliptic problem, so that the expectation of the total potential energy of the problem is minimised. It is referred to as the basic variational low-rank approach with successive rank-one updates (VLR-SR1U scheme), and is comparable to a special scheme in [158]. The resulting low-rank approximation is suboptimal in the sense that it is not the minimiser of the expectation of the mentioned energy. A corresponding optimisation scheme — the variational low-rank optimisation (VLR-OPT) scheme — is derived, which may be used as an independent tool to optimise an approximation of fixed rank. It extends the basic VLR-SR1U scheme to optimise a current approximation on the fly (during a rank-one update) or not until a final rank is reached. Additionally, the residual-based solution space estimator (RBSSE scheme) is introduced to rate stochastic basis functions not yet used for their ability to describe the solution (the idea is comparable to [143]). That scheme also extends the basic VLR-SR1U scheme

to enable an adaptive construction of the solution space. The contributed schemes are in detail presented in Section 2.7.2.

The numerical behaviour of all schemes is discussed in the context of a stochastic stationary groundwater flow problem in Section 4.1.4. There, the basic VLR-SR1U scheme and its extension by the VLR-OPT scheme are compared to direct integration schemes like (Q)-MC methods and the Smolyak algorithm in different configurations. The VLR-SR1U schemes are efficient, they converge faster than (Q)-MC methods, but cannot beat the best configuration of the Smolyak algorithm. Furthermore, the ability of the RBSSE scheme to choose relevant stochastic basis functions for the description of the solution is demonstrated. The basic VLR-SR1U scheme and the VLR-OPT scheme are also applied to a laminated composite structure with a linear constitutive law and an uncertain material, see Section 4.2.2.4. However, for an accurate approximation in the considered solution space a relatively high rank is required. A flexible generic implementation for the schemes in the context of a component-based software architecture is presented in Section 3.4.5.

1.2 Software Reuse in the Simulation of Probabilistic Models

The development of software systems exclusively from scratch meets the requirements of computational sciences less and less. The reuse of software [135, 187, 64, 176] is a promising approach to keep complexity within manageable bounds, see Section 3.3.1. The concept of software components [45, 197, 97, 106, 114, 93] identifies a systematic reuse of software, which encourages the development of maintainable and extensible — and consequently sustainable — software systems, see Section 3.3.3. A software component may be understood as a software unit consisting of an interface — the component interface (CI) — and an implementation and is exclusively specified by its CI. In this thesis a software component is interchangeable and may be distributed. The distribution enables one to exploit the hardware resources in a heterogeneous environment for a potentially extensive computation. It comes along with a late binding of a component, namely at runtime. As a consequence, components can be interchanged at runtime, which increases the flexibility inside the software system. The third-party software solutions for uncertainty quantification in [58, 189, 165, 86, 53, 171, 120] do not follow a component-based approach. Another concept for software reuse is the generic programming [144, 45, 180], see Section 3.3.2. It is here understood as a programming paradigm to implement an

algorithm on the basis of generic types. For a compilation the generic types are specialised by concrete types. In this manner, the same implementation of the algorithm is used for different concrete types, between which an inheritance relation does not need to exist. While generic programming happens on the implementation level, distributed software components put their emphasis on the runtime level. The idea of distributed generic software components combines the mentioned concepts through the possibility to declare generic CIs [40, 162, 161, 160], see Section 3.3.4. A generic CI is one with generic type declarations. This idea is recent and probably not widely known, as only a few component-based architectures support generic CIs [151, 162, 40].

The simulation of probabilistic models involves the simulation or at least the construction of deterministic models. The simulation of deterministic models is not a recent discipline, and usually addressed by a number of well-engineered third-party codes. Such simulation codes are preferably used in the context of this thesis. On the one hand, a software system for the simulation of probabilistic models should be quite general, matching a wide variety of problems to be solved. On the other hand, third-party simulation codes for deterministic models may be quite individual.

The content of this section is extensively discussed in Chapter 3.

1.2.1 Contributions of this Thesis

A software system for the simulation of probabilistic models is proposed here, which enables the use of third-party simulation codes for deterministic models, but is not fixed wired to a special one of these codes. Quite the contrary, the software system receives the application-specific semantics of the probabilistic model at the latest possible instant of time, namely at runtime. As a consequence, the software system is configured at runtime towards a concrete probabilistic model and a corresponding (third-party) simulation code for deterministic models without the requirement of recompilation. The general appearance of a simulation code for a deterministic model is analysed in the context of different numerical schemes for the simulation of probabilistic models, and is then described by a component interface corresponding to a distributed generic software component, see Section 3.4.4. The binding to a concrete implementation at runtime induces the application-specific semantics to the software system. Probabilistic models of different application-specific subjects are simulated here by the proposed software system. For each probabilistic model a different third-party simulation code for deterministic models is used, see Section 3.4.4.2.

Distributed generic software components are the main architectural concept of the software system and used for many concerns other than the simulation of a deterministic model. These concerns are reflected either by single components or compositions of components. A number of components are reused in different contexts and may be instantiated multiple times for a distributed computation. Some components may also be reused beyond the simulation of probabilistic models, see Section 3.4.3. The concept of distributed generic software components is identified here to enable a transfer of the generic support from the implementation level to the runtime level, which maybe would not have been expected, see Section 3.3.4.

Numerical schemes for the simulation of probabilistic models are developed and implemented on the basis of generic types. The implementation enables a quite flexible configuration. The configuration can be done in different concerns separately, so that the implementation can be adapted to a special heterogeneous runtime environment or to special properties of the system to be solved, so that these properties are efficiently handled, see Section 3.4.5.

The proposed architecture is already introduced in [109, 108, 110].

1.3 Uncertainty in Laminated Composite Plates

Laminated composite plates consists of jointly glued layers and are widespread nowadays. Their location-dependent uncertain characteristics are a challenge to accurate computer-aided simulation. These uncertain characteristics like Young's modulus, the shear modulus, and Poisson's ratio were recently described by random fields. In [155, 39, 148, 43] Gaussian distributed random fields are chosen for the description of those characteristics. There, the SGM is applied to simulate the associated problem, the MC method may be used for a comparison. It is mostly not clear to which extent measurement data enter the uncertainty descriptions in these works. A parameterised model for a non-Gaussian positive-definite matrix-valued random field is presented in [193], which describes an in general inhomogeneous anisotropic material for an elliptic problem. The corresponding parameters are identified through experimental data. In [139], samples of the uncertain Young's modulus are obtained through experimental data, which result from a set of loaded structures of identical kind and identical geometrical dimensions. These samples are used in [138] to construct a non-Gaussian random field for Young's modulus represented by a truncated KLE. The random variables of the KLE are, at this, discretised by truncated PCEs, where the PC coefficients are determined through an identification process.

Alternatively, the composite material can be directly observed through photographic images of test specimens which are not externally loaded. A usually non-Gaussian mathematical description for the uncertain composite material is then derived from these images. This is done in [18, 76, 31] through the *moving-window technique* (*MWT*), with which a rectangular window of fixed size is shifted over a given image to scan the composite material. The scanned material at a current window position is mapped onto a finite element model of the same size as the window, and a sequence of load tests identifies the essential material properties for that model. The material is assumed to be homogeneous, and as a consequence each window scan leads to a sample of the uncertain material properties. Statistics can be obtained on the basis of these samples.

None of the above publications deal with a fully anisotropic and uncertain local material tensor of the composite material in a three-dimensional geometrical domain. This is, however, done for a laminated carbon-fibre-reinforced composite material in [94], in which distribution and covariance functions for the local material tensor are constructed through the MWT. Additionally, model reduction techniques are applied to handle the extensive amount of arising data. Further publications about that work are in preparation, a summary is also presented in Section 4.2.1.

1.3.1 Contributions of this Thesis

The distribution and covariance functions in [94] for the fully anisotropic and uncertain composite material in a three-dimensional geometrical domain are used to construct non-Gaussian random fields for a description of the composite material. The author of this thesis is intensively involved in that task.

The composite material is formally divided into virtual interlayers. These interlayers cover the geometrical areas, where the assumed sources of uncertainty are located, namely the voids within the matrix (adhesive) and irregular borders between fibre and matrix layers. A non-Gaussian random field for an interlayer is exemplarily constructed from the input statistics (the given distribution and covariance functions). This reveals a computationally extensive task. Numerical experiments demonstrate that a concrete interlayer requires a comparatively large stochastic dimension of up to six hundred to capture an acceptable accuracy. The corresponding random field is expressed by different representations. The representation — which is first of all obtained — is a generator, which is non-linearly dependent on a Gaussian vector. The generator assures the positive-definiteness of the material because it approximates the matrix logarithm of the local material tensor and then applies the matrix

exponentiation. This generator can be simply sampled and is used to obtain the other representations. The second and third representations are provided in the form of a truncated PCE and a truncated KLE. Each of the three representations may be used in any numerical scheme based on sampling like direct integration, projection, or stochastic collocation schemes (as long as the truncated PC and KL representations are positive-definite). The truncated PCE or KLE may additionally be applied within an SGM discretisation.

The exemplarily constructed interlayer is used for the description of the fully anisotropic and partially uncertain material of a laminated composite structure with a linear constitutive law. A delamination between the mentioned interlayer and the overlying layer is additionally specified. The structure is simulated by a basic MC method. The variance of stresses in the area of the delamination is comparably large, which may essentially influence a progression of the delamination. This demonstrates the purpose of an uncertainty quantification. A regression approach is applied to describe the solution in stochastic polynomials up to first order. This projected solution maintains most of the variance and converges faster than the MC method. The basic VLR-SR1U and VLR-OPT schemes are applied for the same stochastic solution space to search for an accurate low-rank approximation. However, they indicate that an accurate approximation requires a relatively high rank.

The modelling of the composite material and the simulation of the composite structure are discussed in Section 4.2.

1.4 Uncertainty in the Preliminary Aircraft Design

In the multidisciplinary field of aircraft design [177] nowadays the main challenges are the reduction of noise, emission and costs, the improvement of transport capacity, engine performance and safety [173], and the possibility of short take-off and landing [175, 83]. Aircraft design consists of several phases which need to be sequentially passed until an aircraft can be manufactured. The preliminary design phase compares various aircraft configurations to search for an optimal design, which is then processed in further phases [89]. Concepts of the preliminary design phase and application cases are, for instance, published in [105, 89, 91] and [190, 81, 188, 90, 211].

The process of aircraft design reveals many stages at which uncertainty enters

[215, 219, 132]. The model of an aircraft or a corresponding component part may be considered for uncertainty quantification. An appropriate probabilistic model may include probabilistic models for subcomponents (for instance the wing box) or subprocesses (for instance the aerodynamic analysis). Statistics of a probabilistic model are often determined through a basic MC method [192, 147, 215, 47, 133, 50]. Alternatively, importance sampling [131, 134], latin hypercube sampling [51], or a first or second order Taylor expansion [77] may be used. The simulation of a deterministic sample of the probabilistic model is usually computationally expensive. As a consequence, response surface methods are frequently applied [50, 133, 134, 104, 131, 47] to obtain computationally less expensive surrogate models. A surrogate model is represented through a regression equation which usually contains linear and quadratic terms of the stochastic input variables as well as corresponding second order interaction terms. The coefficients of the regression equation are computed through the outcomes of a set of performed (or simulated) experiments. The experiments are systematically selected through a *design of experiments* (*DoE*) [142]. The DoE defines for each stochastic input variable a fixed set of levels to which an experiment can be configured. When each level combination between the stochastic input variables is considered, the DoE is a *factorial design*. In contrast, a *fractional factorial design* assumes, that only a fraction of the experiments which results from the factorial design is enough for a solution of acceptable accuracy. Literature reviews about methods for the quantification of uncertainties in the field of aircraft design are published in [219, 221].

The flight of an aeroplane can be divided into the take-off, the cruise, and the landing [16, 88]. At each of these stages the speed of the aeroplane differs. As a consequence, there are different requirements on the wing. During the take-off the speed is comparably low, therefore a wing has to induce a high lift, while the drag should be kept as low as possible to get major acceleration. By contrast, the cruise requires a small drag, and the landing needs a high lift to establish a low speed. Therefore, the shape of the wing needs to be adapted to provide the different physical characteristics. This is obtained through adjustable flaps, which are attached to the wing to adapt its shape [16, 101, 88, 178, 175]. Furthermore, a *blown flap* system may be applied, which blows the propulsion air over the flaps, so that the lift additionally increases [16, 175].

1.4.1 Contributions of this Thesis

A deterministic reference model of a future civil aircraft with an active high-lift device is provided by a cooperation partner. The high-lift device is based on a blown

flap concept combined with fuel-efficient turboprop engines. Here, the robustness of the reference model towards noises is quantified through the introduction of a probabilistic model for the aircraft and its simulation. Parameters of the aircraft design which mainly influence the relevant design characteristics are stochastically described. A basic MC method provides statistics about the probabilistic model and indicates the aircraft model to be robust. An evaluation of the probabilistic model for a sample of the uncertain parameters is very expensive. As a consequence, a surrogate model is constructed which needs less samples of the probabilistic model than the MC method to provide statistics of the same quality. The surrogate model is simply a truncated PC representation, at which the PC coefficients are computed by a regression approach. The modelling and the simulation of the probabilistic aircraft design are presented in Section 4.3.

The simulation of a deterministic aircraft sample is performed by the simulation code PrADO. The component implementation coPrADO is developed to use PrADO inside the frameworks of the distributed generic component-based software architecture. coPrADO was implemented by the author of this thesis and enables one to call many instances of PrADO in a distributed system. It is presented in Section 3.4.4.3.

1.5 The Structure of this Thesis

This thesis is structured in five chapters. The first and last chapters represent the overall introduction and conclusion. Each of the other chapters additionally provides an own conclusion in the last section of the chapter. In some cases, a group of subsections may also be summarised in a separate subsection.

Chapter 1 briefly introduces the fields of interest associated with this thesis. Numerical schemes for the simulation of probabilistic models are addressed in Chapter 2 which also includes the here developed numerical schemes to obtain a low-rank approximation of the solution in an adaptively constructed solution space. Chapter 3 presents the software realisation for the simulation of probabilistic models which is based on distributed generic software components. The numerical experiments are presented in Chapter 4. The modelling and simulation of the laminated composite and the aircraft design — introduced in the last sections — are also discussed there. Chapter 5 summarises the contributions of this thesis and provides an outlook.

Chapter 2

Numerical Schemes Based on Probabilistic Models

This chapter concentrates on numerical schemes for the simulation of probabilistic models, at which the uncertainty inside the models is represented by random fields. The predominant numerical schemes are direct integration schemes [37, 124, 126, 123, 71, 170, 52, 99], projection [145, 122] and stochastic collocation schemes [11, 153, 66, 216, 20, 26, 208, 62], as well as the stochastic Galerkin method (SGM) [73, 128, 12, 217, 127, 158, 54, 130]. The approaches contributed here belong to the latter numerical scheme and focus on a low-rank approximation of the solution with successive rank-one updates optionally in an adaptively constructed solution space [111].

Random fields and an understanding of probabilistic models are presented in Sections 2.1 and 2.2. Direct integration schemes are discussed in Section 2.3. Projection and collocation schemes are outlined in Sections 2.4 and 2.5. The SGM is addressed in Section 2.6. Corresponding successive rank update schemes — to which also the contributed schemes belong — are presented in Section 2.7.

2.1 Random Fields

An uncertain and location-dependent characteristic of a probabilistic model can be represented by a random field [5, 6, 206]

$$\nu : X \times \Omega \to \mathbb{R}, \tag{2.1}$$

where $X \subset \mathbb{R}^d$ is the geometrical space, and Ω is the set of elementary events of the probability space $\mathcal{P} := (\Omega, \mathcal{F}, P)$. $\mathcal{F} \subseteq 2^\Omega$ is the σ-algebra and $P(\omega)$ the probability measure. In extension to Eq. (2.1), the random field may have values in a vector space like in the numerical experiments of Section 4.2.

Series expansions of random fields and their truncations are discussed in Section 2.1.1. Separated representations in general and rank-r approximations for the coefficients of the discretised expansions in particular are addressed in Section 2.1.2.

2.1.1 Expansions and Their Truncations

The *polynomial chaos expansion* (*PCE*) [217, 73, 128, 129, 146] and the *Karhunen-Loève expansion* (*KLE*) [125, 73, 128, 129, 98] are possibilities to represent a random field. The PCE describes the random field in orthogonal polynomials defined on mutually independent random variables, while the KLE is a spectral decomposition of the random field involving uncorrelated random variables. When the random variables of the KLE are dependent ones each random variable may be represented by the PCE to obtain a description in independent random variables. Both the PCE and the KLE are truncated for computational practicality. There are different approaches to choose a finite set of orthogonal polynomials for the truncated PCE. A finite set is here called basic when the choice depends only on an upper bound for the polynomial order. The number of polynomials in such a set — and consequently the number of terms within the truncated PCE — grows rapidly when the stochastic dimension or the upper bound for the polynomial order is increased. Many of the involved polynomials may not have a significant contribution in describing the random field so that they could actually be neglected. More sophisticated approaches try to find a set with few polynomials which describe the random field significantly (or at least far less polynomials than a basic finite set contains). A corresponding truncated PCE may be addressed under the term of a sparse polynomial chaos expansion [201, 26, 54, 56, 23, 66, 153, 111, 143, 87].

The PCE and the KLE are discussed in Sections 2.1.1.1 and 2.1.1.2. Basic finite sets of orthogonal stochastic polynomials are presented in Section 2.1.1.3, and sparse PCEs are considered in Section 2.1.1.4. Finally, an analytical transfer from the truncated KLE of a Gaussian distributed field to the PCE of a lognormally distributed field is derived in Section 2.1.1.5. (The latter is used in the numerical experiments in Section 4.1.)

2.1.1.1 Polynomial Chaos Expansion

An arbitrary random field ν of finite variance can be described through a PCE [217, 73, 128, 129, 146]

$$\nu(\boldsymbol{x},\omega) \;:=\; \sum_{\alpha\in\mathcal{I}} \nu_\alpha(\boldsymbol{x})\;\psi_\alpha(\boldsymbol{\theta}(\omega)), \tag{2.2}$$

which converges in the L_2 sense. The orthogonal polynomials $\{\psi_\alpha\}_{\alpha\in\mathcal{I}}$ are defined on a random vector $\boldsymbol{\theta} = (\theta_1, \dots, \theta_m, \dots)$ consisting of mutually independent random variables. ν_α is the geometrical coefficient function corresponding to polynomial ψ_α. $\mathcal{I} := \{\alpha \in \mathbb{N}_0^{(\mathbb{N})} \mid |\alpha|_1 < \infty\}$ is a set of multi-indices, where only finitely many integers of a multi-index $\alpha := (\alpha_1, \dots, \alpha_m, \dots)$ are non-zero and the *modulus* of a multi-index is defined as

$$|\alpha|_1 \;:=\; \sum_{i=1}^{\infty} \alpha_i. \tag{2.3}$$

The *factorial* of a multi-index is introduced by

$$\alpha! \;:=\; \prod_{i=1}^{\infty} \alpha_i!. \tag{2.4}$$

The use of a multi-index to indicate a multi-dimensional polynomial (and the corresponding coefficient function) is traced back to the common construction of the polynomial: it results from the product of one-dimensional polynomials, which are defined in the single random variables of $\boldsymbol{\theta}$:

$$\psi_\alpha(\boldsymbol{\theta}(\omega)) \;=\; \prod_{i=1}^{\infty} \psi_{\alpha_i}(\theta_i(\omega)). \tag{2.5}$$

Thus, the multi-index indicates the one-dimensional polynomials used for each dimension. (There is usually a one-to-one correspondence between the indices of a multi-index and the orders of the one-dimensional polynomials. That is also assumed here.)

Originally, the orthogonal stochastic polynomials were Hermite polynomials defined on Gaussian random variables [212]. An exponential convergence of the appropriate PCE could be demonstrated in [217] for Gaussian fields. This is traced back to the fact that the weight function — with respect to which the polynomials are orthogonal — is exactly the same as the probability density function of the Gaussian random

variables. However, the convergence of the mentioned PCE may be significantly slower for non-Gaussian fields [217]. As a consequence of this, other polynomials were associated with other random variables so that the probability density function of the random variables matches the weight function of the orthogonality relation of the polynomials; an example are Legendre polynomials and uniformly distributed variables. Such polynomials in combination with their associated random variables are discussed in [217]. At this, the broader term *generalised polynomial chaos expansion* [218] may be used, but here it is simply called PCE.

When Eq. (2.2) is multiplied by ψ_β ($\beta \in \mathcal{I}$) and integrated over Ω, then the orthogonality relation of the polynomials leads to the formula to compute the coefficient function ν_β:

$$\nu_\beta(\boldsymbol{x}) \;=\; \frac{\mathbb{E}\left(\nu(\boldsymbol{x},\omega)\;\psi_\beta(\boldsymbol{\theta}(\omega))\right)}{\mathbb{E}\left(\psi_\beta^2\right)} \tag{2.6}$$

with the expectation operator $\mathbb{E}(\cdot) := \int_\Omega \cdot \,\mathrm{d}P(\omega)$.

For practical issues, a PCE is discretised through a truncation. The truncation leads to finitely many terms, which are indexed by the set $\mathcal{I}^c$ of multi-indices. c means the restriction rule (or condition) under which the truncation is obtained. The truncated PCE provides an approximation for ν:

$$\nu(\boldsymbol{x},\omega) \;\approx\; \nu_c(\boldsymbol{x},\omega) \;:=\; \sum_{\alpha\in\mathcal{I}^c} \nu_\alpha(\boldsymbol{x})\;\psi_\alpha(\boldsymbol{\theta}(\omega)). \tag{2.7}$$

The finite set $\mathcal{I}^c$ restricts the stochastic polynomials to a dimension of m ($m < \infty$), so that $\mathcal{I}^c \subset \mathbb{N}_0^m$. Some basic finite sets are presented in Section 2.1.1.3, and more sophisticated ones are discussed in Section 2.1.1.4.

When the truncated PCE in Eq. (2.7) is additionally discretised in the geometrical space by finite element basis functions $\{\phi_i\}_{i\in\{1,\ldots,N_X\}}$ then the fully discretised random field can be written as

$$\nu_c^h(\boldsymbol{x},\omega) \;:=\; \sum_{\alpha\in\mathcal{I}^c}\sum_{i=1}^{N_X} \nu_{i,\alpha}\phi_i(\boldsymbol{x})\psi_\alpha(\boldsymbol{\theta}(\omega)). \tag{2.8}$$

The coefficients $\{\nu_{i,\alpha}\}_{i\in\{1,\ldots,N_X\},\alpha\in\mathcal{I}^c}$ may be represented in form of a vector $\boldsymbol{w} \in \mathbb{R}^{N_X\cdot N_S}$ or a matrix $\boldsymbol{W} \in \mathbb{R}^{N_X\times N_S}$ with $N_S := |\mathcal{I}^c|$ (cardinality of $\mathcal{I}^c$). In that case, the geometrical basis functions and the stochastic polynomials are composed in

vectors $\boldsymbol{\phi} := (\phi_1, \ldots, \phi_{N_X})^T$ and $\boldsymbol{\psi} := (\psi_{\alpha_1}, \ldots, \psi_{\alpha_{N_S}})^T$, so that Eq. (2.8) can be restated as

$$\begin{aligned} (2.9) \qquad & \nu_c^h(\boldsymbol{x}, \omega) \;=\; (\boldsymbol{\psi} \otimes \boldsymbol{\phi})^T \, \boldsymbol{w} \;=\; \boldsymbol{\phi}^T \, \boldsymbol{W} \, \boldsymbol{\psi} \qquad \text{with} \\ & \boldsymbol{w} \;:=\; (\nu_{1,\alpha_1}, \ldots, \nu_{N_X,\alpha_1}, \;\; \cdots \;\; , \nu_{1,\alpha_{N_S}}, \ldots, \nu_{N_X,\alpha_{N_S}})^T, \\ & \boldsymbol{W} \;:=\; \begin{pmatrix} \nu_{1,\alpha_1} & \cdots & \nu_{1,\alpha_{N_S}} \\ \vdots & \ddots & \vdots \\ \nu_{N_X,\alpha_1} & \cdots & \nu_{N_X,\alpha_{N_S}} \end{pmatrix}. \end{aligned}$$

That means that $\boldsymbol{w}$ results from stacking all columns of $\boldsymbol{W}$ in sequence to a vector.

2.1.1.2 Karhunen-Loève Expansion

The KLE [125, 73, 128, 129, 98] describes a random field ν in a spectral representation

$$(2.10) \qquad \nu(\boldsymbol{x}, \omega) \;:=\; \bar{\nu}(\boldsymbol{x}) \;+\; \sum_{i=1}^{\infty} \sqrt{\lambda_i} \, \nu_i(\boldsymbol{x}) \, \xi_i(\omega).$$

$\bar{\nu}$ is the expected value of ν. The random variables $\{\xi_i\}_{i \in \mathbb{N}}$ are uncorrelated and centred with unit variance. The λ_i's are the decreasingly ordered eigenvalues of the Fredholm eigenproblem [128, 129, 98]

$$(2.11) \qquad \underbrace{\int_X cov_\nu(\boldsymbol{x}, \boldsymbol{y}) \, \nu_i(\boldsymbol{y}) \, \mathrm{d}\boldsymbol{y}}_{=:(A\nu_i)(\boldsymbol{x})} \;=\; \lambda_i \, \nu_i(\boldsymbol{x}),$$

where the ν_i's are the corresponding eigenfunctions and cov_ν is the covariance function of ν. The eigenvalues are non-negative and have a decreasing order, as the operator A is symmetric, positive semi-definite, and compact on $L_2(X)$. The KLE best approximates the variance with the fewest terms in comparison to all other expansions of the random field [129].

When ν is a Gaussian field the random variables $\{\xi_i\}_{i \in \mathbb{N}}$ are Gaussian as well [128]. Handling with Gaussian random variables is practical, as uncorrelated automatically means independent, and consequently Fubini's theorem [191] is applicable.

To get the same applicability for uncorrelated but dependent random variables $\{\xi_i\}_{i \in \mathbb{N}}$, the PCE can be used to expand each ξ_i in orthogonal polynomials defined

in independent random variables: $\xi_i(\omega) := \sum_{\alpha\in\mathcal{I}\setminus\{\mathbf{0}\}} \xi_{i,\alpha}\ \psi_\alpha(\boldsymbol{\theta}(\omega))$ (the constant polynomial identified by multi-index $\mathbf{0} := (0,\dots,0,\dots)$ can be left out because the ξ_i's are centred). Then, Eq. (2.10) becomes

$$\nu(\boldsymbol{x},\omega) \;=\; \bar{\nu}(\boldsymbol{x}) \;+\; \sum_{i=1}^{\infty} \sqrt{\lambda_i}\ \nu_i(\boldsymbol{x}) \sum_{\alpha\in\mathcal{I}\setminus\{\mathbf{0}\}} \xi_{i,\alpha}\ \psi_\alpha(\boldsymbol{\theta}(\omega))\ . \tag{2.12}$$

When the PCE $\sum_{\alpha\in\mathcal{I}} \nu_\alpha(\boldsymbol{x})\ \psi_\alpha(\boldsymbol{\theta}(\omega))$ for ν is additionally given as an input [98] the coefficients $\{\xi_{i,\alpha}\}_{\alpha\in\mathcal{I}\setminus\{\mathbf{0}\}}$ for the i'th KL term can be computed by

$$\xi_{i,\alpha} \;=\; \frac{1}{\sqrt{\lambda_i}} \int_X \nu_\alpha(\boldsymbol{x})\ \nu_i(\boldsymbol{x})\ \mathrm{d}\boldsymbol{x}. \tag{2.13}$$

For practical issues, a KLE is truncated:

$$\nu(\boldsymbol{x},\omega) \;\approx\; \nu_m(\boldsymbol{x},\omega) := \bar{\nu}(\boldsymbol{x}) \;+\; \sum_{i=1}^{m} \sqrt{\lambda_i}\ \nu_i(\boldsymbol{x})\ \xi_i(\omega). \tag{2.14}$$

When the uncorrelated random variables are discretised through truncated PCEs with stochastic polynomials identified by the set $\mathcal{I}^c$ and the geometrical functions — the expectation and the eigenfunctions — are discretised through finite elements with N_X geometrical basis functions, Eq. (2.14) results in the fully discretised KLE [98]

$$\nu_m^h(\boldsymbol{x},\omega) := \sum_{j=1}^{N_X} \bar{\nu}_j\ \phi_j(\boldsymbol{x}) + \sum_{i=1}^{m} \sqrt{\lambda_i} \sum_{j=1}^{N_X} \nu_{i,j}\ \phi_j(\boldsymbol{x}) \sum_{\alpha\in\mathcal{I}^c\setminus\{\mathbf{0}\}} \xi_{i,\alpha}\ \psi_\alpha(\boldsymbol{\theta}(\omega)), \tag{2.15}$$

where $\boldsymbol{\nu_i} \;=\; (\nu_{i,1},\dots,\nu_{i,N_X})^T$ is the eigenvector corresponding to eigenvalue λ_i. The eigenvalues and eigenvectors are obtained when solving the symmetric general eigenvalue problem

$$\boldsymbol{MCM}\ \boldsymbol{\nu}_i \;=\; \lambda_i\ \boldsymbol{M}\ \boldsymbol{\nu_i}, \tag{2.16}$$

which results from a finite element discretisation of the eigenproblem in Eq. (2.11). $\boldsymbol{C}$ is the corresponding covariance matrix and $\boldsymbol{M}$ the mass matrix. When an appropriately discretised PCE ν_c^h with coefficient vectors $\{\boldsymbol{\nu_\alpha}\}_{\alpha\in\mathcal{I}^c}$ is available as an input [98] the coefficients $\{\xi_{i,\alpha}\}_{\alpha\in\mathcal{I}^c\setminus\{\mathbf{0}\}}$ can be computed by

$$\xi_{i,\alpha} \;\approx\; \frac{1}{\sqrt{\lambda_i}}\ \boldsymbol{\nu}_\alpha^T\ \boldsymbol{M}\ \boldsymbol{\nu}_i. \tag{2.17}$$

This formula results from a finite element discretisation of Eq. (2.13).

The discretised KLE in Eq. (2.15) can be rewritten in a vector- and matrix-based notation. For this purpose the eigenvalues are combined with the geometrical coefficients: $\tilde{\nu}_{i,j} := \sqrt{\lambda_i}\ \nu_{i,j}$. For each eigenvalue these weighted coefficients are composed to a vector $\tilde{\boldsymbol{\nu}}_i := (\tilde{\nu}_{i,1}, \ldots, \tilde{\nu}_{i,N_X})^T$ ($i \in \{1, \ldots, m\}$) and analogously for the stochastic coefficients: $\boldsymbol{\xi}_i := (\xi_{i,\alpha_1}, \ldots, \xi_{i,\alpha_{N_S}})^T$. The coefficients of the discretised expectation and the geometrical basis functions and stochastic polynomials are also treated in this manner: $\bar{\boldsymbol{\nu}} := (\bar{\nu}_1, \ldots, \bar{\nu}_{N_X})^T$, $\boldsymbol{\phi} := (\phi_1, \ldots, \phi_{N_X})^T$, $\boldsymbol{\psi} := (\psi_{\alpha_1}, \ldots, \psi_{\alpha_{N_S}})^T$. Then, Eq. (2.15) becomes

$$\nu_m^h(\boldsymbol{x}, \omega) = \bar{\boldsymbol{\nu}}^T\, \boldsymbol{\phi} + \sum_{i=1}^{m} \boldsymbol{\phi}^T\, \tilde{\boldsymbol{\nu}}_i\, \boldsymbol{\xi}_i^T\, \boldsymbol{\psi} \tag{2.18}$$

$$= \bar{\boldsymbol{\nu}}^T\, \boldsymbol{\phi} + \boldsymbol{\phi}^T\, \widetilde{\boldsymbol{Y}}\, \boldsymbol{Z}^T\, \boldsymbol{\psi}, \tag{2.19}$$

$$= \bar{\boldsymbol{\nu}}^T\, \boldsymbol{\phi} + \boldsymbol{\phi}^T\, \boldsymbol{Y}\, \boldsymbol{\Lambda}\, \boldsymbol{Z}^T\, \boldsymbol{\psi}. \tag{2.20}$$

The i'th column of $\widetilde{\boldsymbol{Y}}$ is given by $\tilde{\boldsymbol{\nu}}_i$ and the i'th column of $\boldsymbol{Z}$ by $\boldsymbol{\xi}_i$. In Eq. (2.20) the matrix $\boldsymbol{\Lambda}$ is a diagonal matrix storing $\sqrt{\lambda_i}$ in its i'th diagonal entry, so that the i'th column of $\boldsymbol{Y}$ is specified by $\boldsymbol{\nu}_i$.

2.1.1.3 Basic Finite Sets of Orthogonal Stochastic Polynomials

Some basic finite sets of orthogonal stochastic polynomials — which may be used for a truncated PCE — are considered in this section through concrete realisations for $\mathcal{I}^c$. These are the sets of multi-indices

$$\mathcal{I}^{max} := \{\alpha \in \mathbb{N}_0^m : \max_{1 \le i \le m} \alpha_i \le p\}, \tag{2.21}$$

$$\mathcal{I}^{mod} := \{\alpha \in \mathbb{N}_0^m : |\alpha|_1 \le p\} \tag{2.22}$$

with parameter p. Thus, the polynomials indexed by $\mathcal{I}^{max}$ are of maximum order $m \cdot p$, whereas the ones indexed by $\mathcal{I}^{mod}$ are of maximum order p. Table 2.1 shows the number of multi-indices $N_S = |\mathcal{I}^{mod}|$ — where operator $|\cdot|$ specifies the cardinality — for the set $\mathcal{I}^{mod}$, when both the stochastic dimension m and the maximum polynomial order p are varied. N_S linearly increases with m for $p = 1$, but for higher maximum polynomial orders the increase grows rapidly, see Figure 2.1. This is even far more dramatic for the set $\mathcal{I}^{max}$. The parameter p alone is obviously not a good choice to restrict the set of multi-indices. More sophisticated choices are discussed in Section 2.1.1.4.

p \ m	5	10	15	20
1	6	11	16	21
2	21	66	136	231
3	56	286	816	1,771
4	126	1,001	3,876	10,626
5	252	3,003	15,504	53,130

Table 2.1: *Basic finite sets of orthogonal stochastic polynomials indexed by $\mathcal{I}^{mod}$: the growth of $|\mathcal{I}^{mod}|$ is presented, when the dimension m and the maximum polynomial order p are varied.*

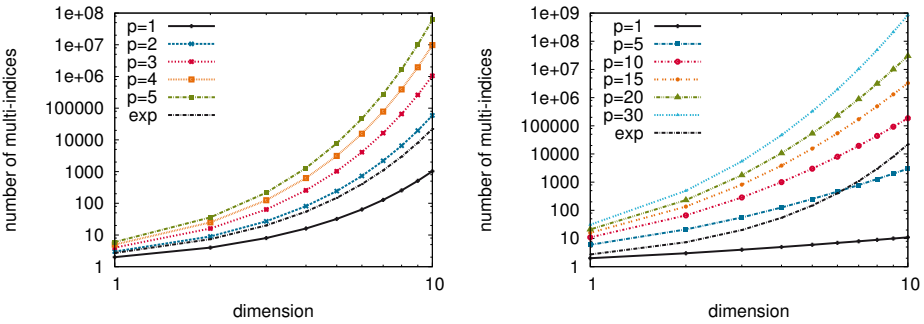


Figure 2.1: *Basic finite sets of orthogonal stochastic polynomials: the growth of $|\mathcal{I}^{max}|$ and $|\mathcal{I}^{mod}|$ is presented in Figures (a) and (b), when the dimension m is increased for different settings of the parameter p. The exponential function is displayed for comparison purposes.*

2.1.1.4 Sparse Polynomial Chaos Expansion

When a random field has an inner sparse structure, and an approximating PCE is constructed for a basic finite set of stochastic polynomials like in Section 2.1.1.3, then many coefficients — and consequently terms — of that PCE may be insignificant. In that case, the relevant PC terms construct a so-called sparse PCE. Definitely, a sparse PCE is a means against the rapid growth in the number of terms inherent in a "basic"

truncated PCE, when the stochastic dimension or the maximum polynomial order is increased. A number of contributions are presented in the literature, non-adaptive and adaptive ones. The next passage discusses non-adaptive contributions [201, 26]. The remaining passages consider the adaptive approaches and divide between an a priori [54, 23] and an a posteriori adaption [24, 25, 26, 56, 66, 154, 153, 111, 143, 87] (the a priori adaptive approaches can be found in the second next passage).

A non-adaptive sparse PCE is introduced for stochastic elliptic problems in [201] and described by the set of multi-indices

$$\mathcal{I}^{\mu,\upsilon} := \left\{ \alpha \in \mathbb{N}_0^m \;\middle|\; |\alpha|_1 \leq \mu \;\wedge\; \sum_{i=1}^{m} \chi_{\mathbb{N}}(\alpha_i) \leq \upsilon \right\} \tag{2.23}$$

with the characteristic function $\chi_{\mathbb{N}}$. The sum of indices per multi-index is restricted to μ; the number of non-zero indices per multi-index is restricted to υ. However, when the parameters μ and υ and additionally the stochastic dimension m increases the number of multi-indices grows quickly as well [23]. Another sparse set of multi-indices is used in [26] and defined by

$$\mathcal{I}^{p,q} := \left\{ \alpha \in \mathbb{N}_0^m \;\middle|\; |\alpha|_q := \left(\sum_{i=1}^{m} \alpha_i^q \right)^{\frac{1}{q}} \leq p \right\} \tag{2.24}$$

with $0 < q < 1$. This set may be denoted as a *hyperbolic* one. That denotation is motivated by the hyperbolic shape, which results when the multi-indices are plotted in an m-dimensional space and parameter q is decreased.

In [54] the PCE for the solution of a stochastic elliptic problem is computed on a coarse geometric mesh to identify the stochastic polynomials of a sparse PCE. This set is then a priori used to obtain the solution on a fine mesh. Such a coarsening can be virtually performed until one geometrical degree of freedom (DoF) remains, that means the geometrical space so to speak disappears. That concept is introduced in [23], the resultant purely stochastic model is called “zero-dimensional model”. A sparse PCE for the solution of that model is determined. The corresponding stochastic polynomials are then used to a priori describe the solution of the actual problem. The initial set of multi-indices — from which few are finally chosen — is $\mathcal{I}^{\mu,\upsilon}$. In both publications the SGM is used to discretise the problems (see Section 2.6).

The next adaptively constructed sparse PCEs are located in the context of stochastic collocation schemes (see Section 2.5). First, the approaches are discussed, which are (or can be) assigned to a regression approach (see Section 2.5.2). Then, approaches

are shortly mentioned which are associated with a sparse grid collocation. A sparse PCE is iteratively and incrementally built up in [56] through a residual-based rating of the stochastic polynomials. Initially, the set of polynomials for the description of the solution only contains the constant polynomial. In each iteration the remaining stochastic polynomials — which are not yet used — are rated. The one which indicates the most contribution for a more accurate description of the solution is added to the set of used stochastic polynomials. Then the problem is solved to obtain the updated solution. Such iterations are performed until the estimated error falls below a threshold. Another a posteriori adapted regression approach is published in [24]. In each iteration of the adaptive process, a "forward" and a "backward" step are performed. During the forward step some remaining stochastic polynomials are one-by-one added to the current set, and each time the corresponding system is solved. A stochastic polynomial with little impact on the solution is discarded. During the backward step the stochastic polynomials — which were chosen in the previous iteration — are discarded one-by-one, and each time the corresponding system is solved as well. When a discard strongly decreases the quality of the solution, the corresponding stochastic polynomial is again added to the current set. Both previous approaches do not adapt the number of collocation points during their processes. An extension of the algorithm in [24] — so that the number of collocation points is adapted — is presented in [25]: while latin hypercube samples are used in [24], nested latin hypercube samples are used in [25]. An improvement of the algorithm in [25] is proposed in [26] to reduce the number of required samples. A similar strategy like in [25] to select stochastic basis polynomials is presented in [66, 153] for the stochastic sparse grid collocation scheme (see Section 2.5.1).

Adaptive residual-based selections of stochastic basis polynomials combined with a low-rank approximation of the solution are used in [111] in the context of the SGM. That approach is a contribution of this thesis and extensively presented in Section 2.7.2. A similar approach is chosen in [143], but a different minimisation problem is solved than in [111] to obtain a low-rank approximation.

The scheme in [87] adaptively constructs a sparse PCE under the assumption that only univariate and bivariate PC terms are of interest.

2.1.1.5 Transformation from a Gaussian to a Lognormal Field

To satisfy well-posedness for a given problem the uncertain parameter may need to be positive. This cannot be directly modelled by a Gaussian field, but a Gaussian field can be transformed to a lognormally distributed field through an exponentiation. An

analytical computation of the coefficients of a PC representation for the lognormal field is in the following derived from a truncated KLE of a Gaussian field as in [72].

Theorem. *A truncated KLE γ_m of a Gaussian field γ is described through*

$$\gamma_m(\boldsymbol{x},\omega) := \bar{\gamma}(\boldsymbol{x}) \; + \; \sum_{i=1}^{m} \sqrt{\lambda_i}\, \gamma_i(\boldsymbol{x})\, \theta_i(\omega), \tag{2.25}$$

where the random variables are mutually independent Gaussian ones. It can be transformed to a lognormally distributed field κ by the exponential function

$$\kappa(\boldsymbol{x},\omega) := e^{\gamma_m(\boldsymbol{x},\omega)}. \tag{2.26}$$

When κ is represented by a PCE

$$\kappa(\boldsymbol{x},\omega) = \sum_{\alpha\in\mathcal{I}} \kappa_\alpha(\boldsymbol{x})\, H_\alpha(\boldsymbol{\theta}(\omega)), \tag{2.27}$$

then the PC coefficients $\{\kappa_\alpha(\boldsymbol{x})\}_{\alpha\in\mathcal{I}}$ in Eq. (2.27) can be analytically computed by

$$\kappa_\alpha(\boldsymbol{x}) = \frac{e^{\bar{\gamma}(\boldsymbol{x})}\, \prod_{i=1}^{m}\, e^{\frac{1}{2}\left(\sqrt{\lambda_i}\gamma_i(\boldsymbol{x})\right)^2}\, \left(\sqrt{\lambda_i}\,\gamma_i(\boldsymbol{x})\right)^i}{\mathbb{E}\left(H_\alpha^2\right)} \tag{2.28}$$

for non-normalised Hermite polynomials $\{H_\alpha\}_{\alpha\in\mathcal{I}}$ and $\mathbb{E}(\cdot) := \int_\Omega \cdot\, dP(\omega)$ (compare [72]). If the PCE is to be expressed in a set of normalised Hermite polynomials $\{\hat{H}_\alpha\}_{\alpha\in\mathcal{I}}$ ($\forall\alpha\in\mathcal{I} : \hat{H}_\alpha := \frac{1}{\sqrt{\mathbb{E}(H_\alpha^2)}} H_\alpha$) the corresponding coefficients $\{\hat{\kappa}_\alpha\}_{\alpha\in\mathcal{I}}$ can be simply computed by using the coefficients of Eq. (2.28):

$$\kappa(\boldsymbol{x},\omega) = \sum_{\alpha\in\mathcal{I}} \underbrace{\kappa_\alpha(\boldsymbol{x})\, \sqrt{\mathbb{E}\left(H_\alpha^2\right)}}_{=:\hat{\kappa}_\alpha(\boldsymbol{x})}\, \hat{H}_\alpha(\boldsymbol{\theta}(\omega)). \tag{2.29}$$

Proof. Using Eq. (2.25) in Eq. (2.26) and embedding the result in Eq. (2.6) (to compute a PC coefficient) leads to

$$\kappa_\alpha(\boldsymbol{x}) = \frac{\mathbb{E}\left(e^{\bar{\gamma}(\boldsymbol{x}) \,+\, \sum_{i=1}^{m} \sqrt{\lambda_i}\, \gamma_i(\boldsymbol{x})\, \theta_i(\omega)}\, H_\alpha(\boldsymbol{\theta}(\omega))\right)}{\mathbb{E}\left(H_\alpha^2\right)} =: \frac{\tilde{\kappa}_\alpha(\boldsymbol{x})}{\mathbb{E}\left(H_\alpha^2\right)}, \tag{2.30}$$

see [72]. For the Hermite polynomials the denominator is analytically known by $\mathbb{E}\left(H_\alpha^2\right) = \alpha!$ [128, 98]. Consequently, the numerator $\tilde{\kappa}_\alpha$ stays to be analysed. In the

first step the multi-dimensional Gaussian measure $P(\omega)$ is resolved through variables $(\eta_1, \ldots, \eta_m) =: \boldsymbol{\eta}$:

$$\begin{aligned}\tilde{\kappa}_\alpha(\boldsymbol{x}) &= \int_{\mathbb{R}^m} e^{\bar{\gamma}(\boldsymbol{x}) + \sum_{i=1}^m \sqrt{\lambda_i}\,\gamma_i(\boldsymbol{x})\,\eta_i}\, H_\alpha(\boldsymbol{\eta})\, \frac{1}{\sqrt{(2\pi)^m}} e^{-\frac{1}{2}\sum_{i=1}^m \eta_i^2}\, \mathrm{d}\boldsymbol{\eta} \\ &= \frac{1}{\sqrt{(2\pi)^m}} e^{\bar{\gamma}(\boldsymbol{x})} \int_{\mathbb{R}^m} \left(\prod_{i=1}^m e^{\sqrt{\lambda_i}\,\gamma_i(\boldsymbol{x})\,\eta_i - \frac{1}{2}\eta_i^2} \right) H_\alpha(\boldsymbol{\eta})\, \mathrm{d}\boldsymbol{\eta}.\end{aligned}$$

In the second step the multi-dimensional Hermite polynomial H_α is expressed in its tensor product representation of one-dimensional Hermite polynomials $\{h_{\alpha_i}\}_{i\in\{1,\ldots,m\}}$: $H_\alpha(\boldsymbol{\eta}) := \prod_{i=1}^m h_{\alpha_i}(\eta_i)$. The product can be extracted from the integral due to Fubini's theorem [191]:

$$\begin{aligned}&= \frac{1}{\sqrt{(2\pi)^m}} e^{\bar{\gamma}(\boldsymbol{x})} \int_{\mathbb{R}^m} \left(\prod_{i=1}^m e^{\sqrt{\lambda_i}\,\gamma_i(\boldsymbol{x})\,\eta_i - \frac{1}{2}\eta_i^2} \right) \left(\prod_{i=1}^m h_{\alpha_i}(\eta_i) \right) \mathrm{d}\boldsymbol{\eta} \\ &= \frac{1}{\sqrt{(2\pi)^m}} e^{\bar{\gamma}(\boldsymbol{x})} \prod_{i=1}^m \int_{\mathbb{R}} e_i^{\sqrt{\lambda_i}\,\gamma_i(\boldsymbol{x})\,\eta_i - \frac{1}{2}\eta_i^2}\, h_{\alpha_i}(\eta_i)\, \mathrm{d}\eta_i.\end{aligned}$$

Next, the square is completed (in the exponent), and a substitution $\tilde{\eta}_i := \eta_i - \sqrt{\lambda_i}\,\gamma_i(\boldsymbol{x})$ follows (compare [72]):

$$\begin{aligned}&= \frac{1}{\sqrt{(2\pi)^m}} e^{\bar{\gamma}(\boldsymbol{x})} \prod_{i=1}^m \int_{\mathbb{R}} e^{-\frac{1}{2}\left[\left(\eta_i - \sqrt{\lambda_i}\gamma_i(\boldsymbol{x})\right)^2 - \left(\sqrt{\lambda_i}\gamma_i(\boldsymbol{x})\right)^2\right]} h_{\alpha_i}(\eta_i)\, \mathrm{d}\eta_i \\ &= \frac{1}{\sqrt{(2\pi)^m}} e^{\bar{\gamma}(\boldsymbol{x})} \prod_{i=1}^m e^{\frac{1}{2}\left(\sqrt{\lambda_i}\gamma_i(\boldsymbol{x})\right)^2} \int_{\mathbb{R}} e^{-\frac{1}{2}\left(\eta_i - \sqrt{\lambda_i}\gamma_i(\boldsymbol{x})\right)^2} h_{\alpha_i}(\eta_i)\, \mathrm{d}\eta_i \\ &= \frac{1}{\sqrt{(2\pi)^m}} e^{\bar{\gamma}(\boldsymbol{x})} \prod_{i=1}^m e^{\frac{1}{2}\left(\sqrt{\lambda_i}\gamma_i(\boldsymbol{x})\right)^2} \int_{\mathbb{R}} e^{-\frac{1}{2}\,\tilde{\eta}_i^2} h_{\alpha_i}(\tilde{\eta}_i + \sqrt{\lambda_i}\,\gamma_i(\boldsymbol{x}))\, \mathrm{d}\tilde{\eta}_i.\end{aligned}$$

A shifted non-normalised one-dimensional Hermite polynomial $h_n(x+y)$ of order n can be written in a binomial *Appel sequence* [198]: $h_n(x+y) = \sum_{j=0}^n \binom{n}{j} x^j\, h_{n-j}(y)$. Applying this form to the occurring shifted Hermite polynomials $h_{\alpha_i}(\tilde{\eta}_i + \sqrt{\lambda_i}\,\gamma_i(\boldsymbol{x})) = h_{\alpha_i}(\sqrt{\lambda_i}\,\gamma_i(\boldsymbol{x}) + \tilde{\eta}_i)$ simplifies the whole formula, as it leads to integrals of the Hermite polynomials in the Gaussian measure, which

are zero excepting the case of the constant Hermite polynomial h_{α_0} ($j = i$):

$$
\begin{aligned}
&= \frac{1}{\sqrt{(2\pi)^m}} e^{\bar{\gamma}(\boldsymbol{x})} \prod_{i=1}^{m} e^{\frac{1}{2}\left(\sqrt{\lambda_i}\gamma_i(\boldsymbol{x})\right)^2} \cdot \\
&\quad \int_{\mathbb{R}} e^{-\frac{1}{2}\,\tilde{\eta}_i^2} \left(\sum_{j=0}^{i} \binom{i}{j} \left(\sqrt{\lambda_i}\,\gamma_i(\boldsymbol{x})\right)^j h_{\alpha_{i-j}}(\tilde{\eta}_i) \right) \mathrm{d}\tilde{\eta}_i \\
&= e^{\bar{\gamma}(\boldsymbol{x})} \prod_{i=1}^{m} e^{\frac{1}{2}\left(\sqrt{\lambda_i}\gamma_i(\boldsymbol{x})\right)^2} \sum_{j=0}^{i} \binom{i}{j} \left(\sqrt{\lambda_i}\,\gamma_i(\boldsymbol{x})\right)^j \cdot \\
&\quad \int_{\mathbb{R}} \frac{1}{\sqrt{(2\pi)^m}} e^{-\frac{1}{2}\,\tilde{\eta}_i^2} h_{\alpha_{i-j}}(\tilde{\eta}_i)\, \mathrm{d}\tilde{\eta}_i \\
&= e^{\bar{\gamma}(\boldsymbol{x})} \prod_{i=1}^{m} e^{\frac{1}{2}\left(\sqrt{\lambda_i}\gamma_i(\boldsymbol{x})\right)^2} \left(\sqrt{\lambda_i}\,\gamma_i(\boldsymbol{x})\right)^i .
\end{aligned}
$$

□

2.1.2 Separated Representations and Rank-r Approximations

Recently, efficient separated representations for variables or coefficients of random fields were proposed through a finite sum of decompositions [158, 130, 111, 143, 122, 55, 103]. Related publications without an explicit application in probabilistic models are for instance [13, 60, 22, 7]. The next two passages introduce the topic of separated representations, the subsequent passages shortly outline the mentioned publications.

A separated representation [41] for a generic function $f(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n)$ can be written as

$$
f(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n) \;\approx\; \sum_{i=1}^{r} g_{1,i} \otimes \cdots \otimes g_{n,i}\,(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n).
$$

It may be a *proper orthogonal decomposition* (*POD*) [159, 158, 157, 42, 41] or a *proper generalised decomposition* (*PGD*) [159, 42, 41]. An essential property of a POD is as follows: a truncated POD with r terms is optimal with respect to a given norm from some inner product, that means no other decomposition with r terms

is better with respect to the considered norm. As the POD is another name for the singular value decomposition (SVD), the singular vectors are orthogonal with respect to the chosen inner product, which explains the word "orthogonal" in POD. Another name for the SVD is the Karhunen-Loève expansion (KLE), see Section 2.1.1.2.

The PGD is the result of a greedy (locally optimal) algorithm, and therefore neither necessarily globally optimal, nor are the vectors usually orthogonal.

Here a separated representation for the geometric and stochastic coefficients of a discretised PCE $\nu_c^h(\boldsymbol{x},\theta) = \boldsymbol{\phi}^T \boldsymbol{W} \boldsymbol{\psi}$ (compare Eq. (2.9)) is focused as in [158, 130, 111, 143]. In other words, a rank-r approximation for the coefficient matrix $\boldsymbol{W} \in \mathbb{R}^{N_X \times N_S}$ is to be determined

$$\boldsymbol{W} \approx \hat{\boldsymbol{W}} := \sum_{i=1}^{r} \boldsymbol{g}_i \, \boldsymbol{h}_i^T = \boldsymbol{G} \, \boldsymbol{H}^T, \tag{2.31}$$

where $\boldsymbol{g}_i \in \mathbb{R}^{N_X}$ and $\boldsymbol{h}_i \in \mathbb{R}^{N_S}$ are column vectors and $r \leq \min\{N_X, N_S\}$. (The summands are tensors of second order.) Furthermore, the sum may be formulated in a product of the matrices $\boldsymbol{G} \in \mathbb{R}^{N_X \times r}$ and $\boldsymbol{H}^T \in \mathbb{R}^{r \times N_S}$, at which the i'th column of $\boldsymbol{G}$ is specified by $\boldsymbol{g}_i$ and the i'th column of $\boldsymbol{H}$ by $\boldsymbol{h}_i$.

The rank-r representation has two advantages, which are essential for a low-rank representation of $\boldsymbol{W}$, that means $r \ll \min\{N_X, N_S\}$. On the one hand the memory requirement reduces from $N_X \times N_S$ floating point variables to $r \times (N_X + N_S)$ floating point variables. On the other hand a more efficient computational handling is enabled. For instance, a matrix-vector product reduces from N_X scalar products to r scalar products and vector scalings:

$$\hat{\boldsymbol{W}} \boldsymbol{v} = \sum_{i=1}^{r} \boldsymbol{g}_i \, \boldsymbol{h}_i^T \, \boldsymbol{v} \underset{a_i := \boldsymbol{h}_i^T \boldsymbol{v}}{=} \sum_{i=1}^{r} \boldsymbol{g}_i a_i \underset{\boldsymbol{b}_i := \boldsymbol{g}_i a_i}{=} \sum_{i=1}^{r} \boldsymbol{b}_i, \tag{2.32}$$

with $v \in \mathbb{R}^{N_S}$ and $a_i \in \mathbb{R}$. Consequently, only $r \times (N_X + N_S)$ multiplications and $(r-1) \times (N_X + N_S - 1)$ additions are required instead of $N_X \times N_S$ multiplications and $N_X \times (N_S - 1)$ additions. The discretised KLE (compare Eq. (2.19)) is already such a low-rank representation of a random field through tensors of second order under the assumption that the KL eigenvalues decrease quickly.

In [158, 130, 111, 143] a low-rank representation with tensors of second order for a separated representation of the geometric and stochastic coefficients is proposed for the solution of a stochastic elliptic PDE discretised by the SGM. In [158] (see Section 2.7.1) a low-rank approximation of the solution is obtained through a minimisation of the expectation of the total potential energy inherent in the considered

problem. There, successive rank-k updates lead to the approximation of a demanded rank r (or a demanded accuracy). However, a successive rank update is identified to be suboptimal [158], so that additional global updates may be required to reach the demanded accuracy. In [111] a successive rank-one update is derived, which also tries to minimise the expectation of the total potential energy. The proposed algorithm is furthermore applied onto projections of the considered problem to derive an optimisation algorithm for approximations of fixed rank. Additionally, an adaptive construction of the solution space is performed (see Section 2.1.1.4). The approaches in [111] are a contribution of this thesis and explicitly discussed in Section 2.7.2. The combination of a low-rank approach and an adaptive solution space construction is also the focus in [143], however the L_2-norm of the residual is minimised to obtain a low-rank approximation.

In [130] a low-rank representation for the solution is embedded in an iterative scheme to solve the linear system resulting from an SGM discretisation of a stochastic elliptic PDE. The corresponding operator of the system is represented through a sum of tensors. This sum leads to a rank increase of the arising matrices in each iteration. An SVD is applied to reduce the matrices back to a low rank, where the truncation is error driven. In this context other iterative schemes and tensors of second or higher orders are focused in [13, 103], at which [13] performs a truncation to a fixed rank.

In contrast to the rank optimisation schemes proposed in [158, 111], an efficient gradient-based scheme is presented in [60]. In [121, 122] a separated representation for the geometric coefficients of the solution and the responses of samples is used in the context of projection schemes (see Section 2.4). A separated representation of the random variables in a probabilistic model is proposed in [55] to break the curse of dimensionality coming along with a full tensor sample grid.

2.2 Probabilistic Models and Their Simulation

A mathematical model for a physical phenomenon describes the essential properties of the phenomenon, so that the corresponding behaviour can be well predicted. The process of deriving a mathematical model exhibits simplified formulations. Relevant uncertain properties of the phenomenon itself or an uncertainty which arises through the mentioned simplifications can be reflected by a probabilistic model, that means through stochastic descriptions of integral parts.

The probabilistic model may appear differently for an uncertainty quantification, de-

pending on the techniques for its simulation. A probabilistic location-dependent model may be simply understood as a random field $u(\boldsymbol{x}, \omega)$, which can be evaluated for samples of the variables. Then, the model acts like a black box. Schemes for its simulation, which are based on sampling, are for instance direct integration schemes (see Section 2.3), projection schemes (see Section 2.4), and stochastic collocation schemes (see Section 2.5). These schemes are often referred to as non-intrusive ones [145, 26, 66, 20], as they only require an access in the black box manner.

A probabilistic model may also appear as a system on which external actions — the sources/sinks — on the system cause reactions — the solution — of the system [128]. Then, the solution u is to be found, so that the equations

$$\begin{aligned} \mathcal{L}(\boldsymbol{x}, \omega; u) &= f(\boldsymbol{x}, \omega) \quad \text{with} \quad \boldsymbol{x} \in X \backslash \partial X, \\ \mathcal{B}(\boldsymbol{x}, \omega; u) &= b(\boldsymbol{x}, \omega) \quad \text{with} \quad \boldsymbol{x} \in \partial X, \end{aligned} \tag{2.33}$$

almost surely hold [216, 62]. X and ∂X are the geometrical domain and its boundary. $\mathcal{L}$ describes the relation between the uncertain sources/sinks f and the uncertain solution u on $X \backslash \partial X$ under the conditions $\mathcal{B}$ at ∂X. The SGM (see Section 2.6) is applied on that appearance of the probabilistic model and is frequently referred to as an intrusive scheme [145, 26].

An example for the previous understanding of a probabilistic model is provided by a stationary groundwater flow problem with an uncertain hydraulic conductivity [128, 129]. It is described by the stochastic linear elliptic partial differential equation

$$-\nabla_{\boldsymbol{x}} \cdot (\kappa(\boldsymbol{x}, \omega) \nabla_{\boldsymbol{x}} u(\boldsymbol{x}, \omega)) = f(\boldsymbol{x}, \omega), \qquad \boldsymbol{x} \in X \backslash \partial X \tag{2.34}$$

$$n(\boldsymbol{x}) \cdot (\kappa(\boldsymbol{x}, \omega) \nabla_{\boldsymbol{x}} u(\boldsymbol{x}, \omega)) = t(\boldsymbol{x}, \omega), \qquad \boldsymbol{x} \in \partial X_N \tag{2.35}$$

$$u(\boldsymbol{x}, \omega) = u_D(\boldsymbol{x}, \omega), \qquad \boldsymbol{x} \in \partial X_D \tag{2.36}$$

defined onto the bounded geometrical domain X with boundary ∂X. It holds $\partial X = \partial X_D \cup \partial X_N$ and $\partial X_D \cap \partial X_N = \emptyset$. ∂X_D and ∂X_N are the boundaries for the essential (Dirichlet) and natural (Neumann) boundary conditions. The hydraulic head u, hydraulic conductivity κ, essential boundary conditions u_D, natural boundary conditions t, and source/sink term f are assumed to be uncertain and therefore represented by random fields.

2.3 Direct Integration

Statistics for the solution of a probabilistic model may be formulated by integrals, which are then directly approximated by (numerical) integration schemes [128, 99].

The most popular integration schemes for high dimensions are Monte Carlo (MC) [37, 124, 126, 172] and Quasi-Monte Carlo (QMC) methods [37, 172, 123] as well as the Smolyak algorithm [71, 70, 169, 170, 52, 99].

An integral to obtain stochastic moments for the solution $u(\boldsymbol{x}, \omega)$ of a probabilistic model can be written as

$$\int_{\Omega} s(\, u(\boldsymbol{x}, \omega)\,)\, \mathrm{d}P(\omega), \tag{2.37}$$

where the function s is associated with the stochastic moment to be computed. When for example $s(u(\boldsymbol{x}, \omega)) := (u(\boldsymbol{x}, \omega))^i$ for $i \in \mathbb{N}$ holds, then the integral describes the i'th location-dependent absolute stochastic moment. Without loss of generality it is assumed that the uncertainty inside u has been mapped onto independent uniformly distributed random variables $\boldsymbol{\eta}(\omega) := (\eta_1(\omega), \ldots, \eta_d(\omega))$ in $[0, 1]^d$, so that $u(\boldsymbol{x}, \omega) = u(\boldsymbol{x}, \boldsymbol{\eta})$ holds. Then, the integral in Eq. (2.37) is identified as a d-dimensional one. The geometrical space is not essential for the further considerations in this section. To simplify matters the geometrical variable $\boldsymbol{x}$ is discarded, so that only a random variable $u(\boldsymbol{\eta})$ stays. Accordingly, the corresponding integral to be approximated can be reformulated in

$$I_d := \int_{[0,1]^d} \underbrace{s(\, u(\boldsymbol{\eta})\,)}_{=: f(\boldsymbol{\eta})}\, \mathrm{d}\boldsymbol{\eta}. \tag{2.38}$$

A numerical scheme $Q_{l,d}$ to approximate I_d can be generically written as

$$I_d \approx Q_{l,d} f := \sum_{i=1}^{n_{l,d}} w_{i,d} f(\boldsymbol{\eta}_{i,d}). \tag{2.39}$$

At this, the function f is sampled at the points $\{\boldsymbol{\eta}_{i,d}\}_{i \in \{1,\ldots,n_{l,d}\}}$ ($\boldsymbol{\eta}_{i,d} \in [0, 1]^d$), and the significance of each sample-point is expressed by the weight $w_{i,d}$ (the volume of the sample space is here one but may be otherwise formally embedded within the weights). The level (order) l of the numerical scheme identifies the number $n_{l,d}$ of sample-points. In some cases, the level and the number are the same.

Different numerical schemes satisfying Eq. (2.39) are discussed in the following sections. The (Q)MC methods are addressed in Section 2.3.1, then the Smolyak algorithm follows in Section 2.3.2.

2.3.1 (Quasi)-Monte Carlo Methods

MC methods [37, 124, 126, 172] sample the variables of the integral in Eq. (2.38) randomly. The basic MC method simply applies a random number generator for the sample space and computes Eq. (2.39). The weights in Eq. (2.39) are then homogeneously $\frac{1}{n}$, at which $n := n_{l,d}$ (to simplify matters) is the number of sample-points. The MC method is popular due to its attractive properties: the convergence is independent of the dimension; its simple idea results in a robust scheme easy to implement with the opportunity of straight forward parallelisation or remote distribution. Its numerical error ϵ can be estimated for a large n by $\epsilon \approx \frac{\sqrt{\sigma^2} Y_N}{\sqrt{n}}$, at which σ^2 is the (constant) variance of the integrand f and Y_N is a standard Gaussian random variable [37]. It is obvious that σ^2 needs to be bounded. The convergence rate can be summarised as $O(\frac{\sqrt{\sigma^2}}{\sqrt{n}})$. This reveals the disadvantage of the MC method, because it indicates a slow convergence. It may be demonstrated by the associated empirical formula: one digit more accuracy requires one hundred times more sample-points. More sophisticated MC methods aim to reduce the variance of the integrand to accelerate the convergence behaviour [37, 124, 126, 172].

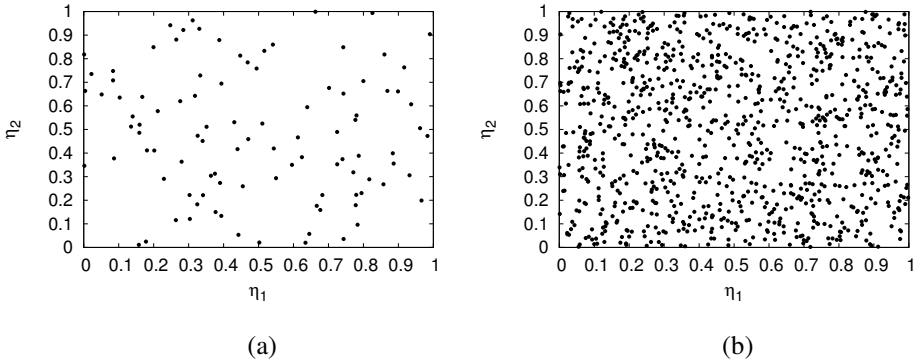


Figure 2.2: *Sample-points of a uniformly distributed random generator: Figure (a) shows* 100 *sample-points, Figure (b) shows* $1,000$ *sample-points.*

Figure 2.2 presents ensembles of sample-points for the basic MC method generated by a uniformly distributed random generator [32]. This ensemble shows a property to which random samples tend: clustering and gaps. A cluster means an area at which many sample-points were generated; a gap means an area at which no sample-points were generated.

QMC methods [149, 150, 37, 172] differ from MC methods in the generation of the sample-points (the weights stay untouched). While MC methods use random number generators QMC methods apply number theoretic point sequences. The idea is to avoid the mentioned clustering and gaps by using points which are more homogeneously spread over the sample space. QMC methods have a convergence rate of $O(\frac{\|f\|_{BV}}{n} (\log n)^d)$, where $\|f\|_{BV}$ is the bounded variation norm of f [99, 37]. Obviously, they are almost independent of the dimension except for a logarithmic factor. Thus, QMC methods may offer a better convergence behaviour at least up to an already high number of dimensions. Furthermore, the convergence depends on the smoothness of the integrand, indicated by factor $\|f\|_{BV}$.

Number theoretic point sequences for QMC methods are, for instance, published by Halton [80] and Sobol [29]. The Halton point sequence is defined by

$$\boldsymbol{\eta}_{i,d} = (\phi_{b_1}(i), \ldots, \phi_{b_d}(i)) \in [0,1]^d, \qquad i = 1, 2, \ldots ,$$

at which $b_1, \ldots, b_d \in \mathbb{N}_{>1}$ are pairwise coprime numbers. b_j is usually chosen to be the j'th prime number. The radical inverse function ϕ_{b_i} constructs a fraction from the number $i = (d_k d_{k-1} \ldots d_1 d_0)_{b_i} = \sum_{j=0}^{k} d_j {b_i}^j$ expressed in the base b_i with $d_j \in \{0, 1, \ldots, b_i - 1\}$ and is given by

$$\phi_{b_j}(i) = (0.d_0 d_1 \ldots d_k)_b = \sum_{j=0}^{k} d_j b^{-1-j}.$$

It is conspicuous that the Halton sequence does not contain the number of sample-points n inside its generation rule. That means: a running QMC sampling by Halton can be flexibly extended to more sample-points when the currently estimated error is not satisfying. The Sobol sequence [29] constructs its sample-points from XOR connections of binary fractions. Figure 2.3 presents ensembles of sample-points corresponding to Halton and Sobol sampling. A recent QMC number generator is published in [123]. There a tensor product of layered two-dimensional grids of sample-points leads to a d-dimensional grid.

Latin Hypercube sampling [136, 172] underlies another idea than the previously discussed (Q)MC samplings. Each dimension of the sample space is partitioned in n segments, which results in n^d cells. A successive elimination of rows and columns finally leads to n sample-points. As a consequence, the sample-point generation is independent of the dimension. When a random input variable has a dominant appearance compared to the others the Latin Hypercube sampling may perform well.

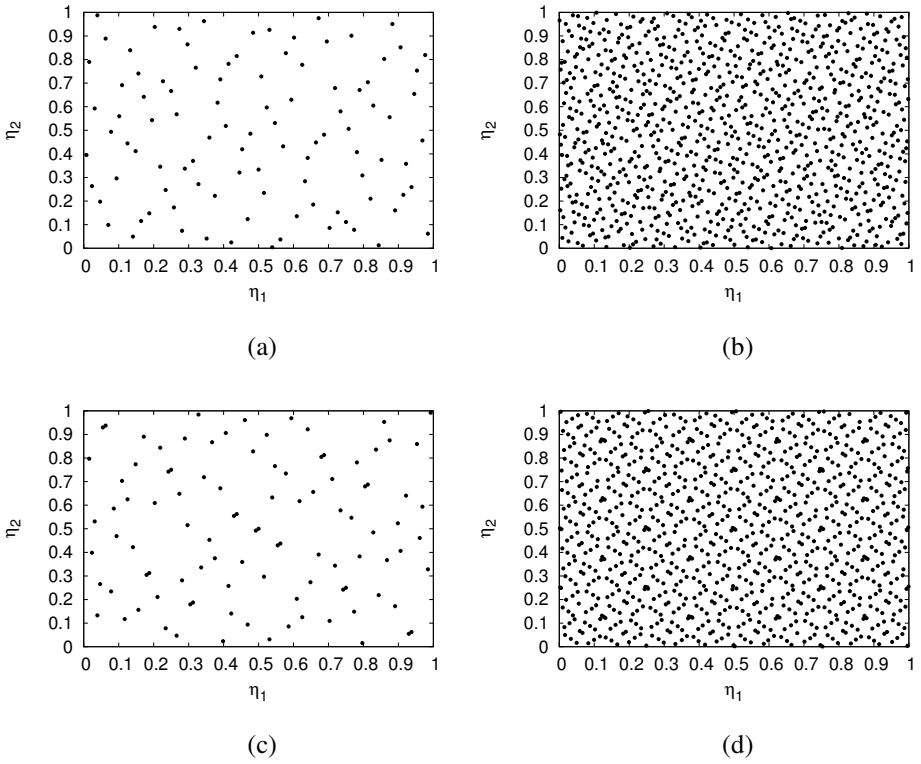


Figure 2.3: *Sample-points of QMC methods: Figures (a) and (b) show* 100 *and* $1,000$ *sample-points obtained by the Halton sequence, Figures (c) and (d) show* 100 *and* $1,000$ *sample-points obtained by the Sobol sequence.*

2.3.2 The Sparse Grid Method

A multi-dimensional quadrature rule may be constructed from one-dimensional ones through a tensor product. A full tensor product suffers from the curse of dimension, that means the number of sample-points grows exponentially when the dimension increases. Sparse tensor products are used by the Smolyak algorithm [71, 70, 169, 170, 52, 99] to counter that curse. Another name for the Smolyak algorithm is "sparse grid method", under which a generalisation is proposed in [71] (however, that name

also refers to the hierarchical interpolation scheme in [36, 35]).

Section 2.3.2.1 addresses some one-dimensional quadrature rules commonly used to construct high-dimensional rules. The Smolyak algorithm is presented in Section 2.3.2.2.

2.3.2.1 One-Dimensional Quadrature Rules

This section presents some one-dimensional quadrature rules [69, 28, 141, 166, 167, 202, 70, 172] and their properties.

A univariate integral I_1 of a function f times a weight function w is formulated by

$$I_1 := \int_{[0,1]} w(\eta) f(\eta) \, \mathrm{d}\eta \tag{2.40}$$

with a uniformly distributed random variable η. A one-dimensional quadrature rule $Q_{k,1} f$ of order $k \in \mathbb{N}$ approximates I_1 by a sum of $n_{k,1}$ many weighted samples of f [69, 116, 28, 172]

$$I_1 \approx Q_{k,1} f := \sum_{i=1}^{n_{k,1}} w_{i,1} f(\eta_{i,1}),$$

where $w_{i,1}$ is the weight corresponding to the i'th sample-point $\eta_{i,1}$.

A quadrature rule may even exactly integrate when f is a polynomial. That property allows to rate the quality of a quadrature rule by the so-called *polynomial degree of exactness* [70, 169, 69], which is the degree of polynomials up to which an integration is exact. Also, if f is a polynomial the integrand may be much different due to w. Consequently, a specific w leads to a class of integrands exactly integrated. Two different kinds of weight functions are regarded here: $w(\eta) := 1$ or $w(\eta) := \frac{1}{\sqrt{2\pi}} e^{-\frac{\eta^2}{2}}$. The latter is the Gaussian measure, the corresponding sample-points are in $(-\infty, \infty)$. When sample-points of the first weight function should be considered in association with a Gaussian measure they can be translated by the inverse of the normal distribution function.

Popular one-dimensional quadrature rules are the Gaussian ones [69, 28, 172]. Their construction is based on a set of orthogonal polynomials associated to the particular w. The sample-points for the $n_{k,1}$-point quadrature rule are then the roots of the orthogonal polynomial of degree $n_{k,1}$. The related degree of exactness is $2n_{k,1} - 1$.

The *Gauss-Legendre* quadrature rule is based on $w(\eta) := 1$ and orthogonal Legendre polynomials. The *Gauss-Hermite quadrature* rule in this thesis is based on the Hermite polynomials with respect to the Gaussian measure.

A pragmatic estimation of a directed error for such a quadrature involves a second quadrature of higher order [71]. It can be defined by the difference formula

$$\Delta_{k+1} := Q_{k+1,1}f - Q_{k,1}f. \tag{2.41}$$

Successive difference formulas along with the definition $Q_{0,1}f := 0$ can be used to get the corresponding higher-order quadrature rule:

$$Q_{k+1,1} = \sum_{i=1}^{k+1} \Delta_i. \tag{2.42}$$

Eq. (2.42) is a so-called *telescoping series* [71], as all elementary summands (excepting the first one $Q_{0,1}$ and the last one $Q_{k+1,1}$) cancel each other out due to the increment by one of index i. This formulation is taken on in the next section.

The significant costs of a quadrature is usually identified by the function evaluations. The estimated error defined in Eq. (2.41) includes two quadratures. Assuming that $Q_{k+1,1}f$ and $Q_{k,1}f$ contains $n_{k,1}+1$ and $n_{k,1}$ sample-points which are all distinct, then a total of $2n_{k,1}+1$ function evaluations are required. Even if more sample-points are involved, the degree of exactness stays. A so-called *nested* quadrature rule [70] of an arbitrary order $k+1$ contains the sample-points of order k next to additional ones. This identifies the sample-points to be so-called nested. At least $n_{k,1}+1$ additional sample-points are necessary to increase the degree of exactness [141]. However, this means a total of $2n_{k,1}+1$ sample-points just as in the previously mentioned non-nested case, but the degree of exactness may be improved. A *Kronrod* quadrature rule [141, 166] is a nested quadrature rule using $n_{k,1}+1$ additional sample-points for an order $k+1$ and increasing the degree of exactness to $3n_{k,1}+1$ when $n_{k,1}$ is even, and $3n_{k,1}+2$ when $n_{k,1}$ is odd. The Kronrod-Patterson quadrature rule [167] extends the Gauss-Legendre quadrature rule to a Kronrod one and uses odd numbers of sample-points. It is proven that a Kronrod extension does not exist for the Gauss-Hermite quadrature rules [95]. Nevertheless, alternatives exist to construct nested quadrature rules also for the Gauss-Hermite one, admittedly with a lower degree of exactness than a Kronrod one [115]. The Clenshaw-Curtis quadrature rule [28, 70] is a further nested rule at which the weight function yields $w(\eta) := 1$. The corresponding sample-points are the extrema of the Chebyshev polynomials of the first kind; the degree of exactness is $n_{k,1}-1$ (here $n_{k,1}$ means all sample-points, not only additional ones). Even though the degree of exactness is less than half of

the one from Gaussian quadrature, in practice Clenshaw-Curtis quadrature is often observed to similarly behave as the Gaussian one [202].

2.3.2.2 Smolyak Algorithm

The Smolyak algorithm [71, 70, 169, 170, 52, 99] constructs a sparse tensor product of one-dimensional quadrature rules so that higher order quadrature is applied in few dimensions while at the same time low order quadrature is applied in most dimensions. In this manner the curse of dimension associated with a full tensor quadrature is addressed.

The difference consideration of one-dimensional quadrature formulas in Eq. (2.42) can be transferred to d-dimensional integration schemes. Then, the tensor product of the mentioned difference formulas are summed up for different levels of the quadrature [71]:

$$I_d \approx Q_{k,d} f = \sum_{\boldsymbol{k} \in \mathcal{I}} (\Delta_{k_1} \otimes \cdots \otimes \Delta_{k_d}) f. \tag{2.43}$$

The i'th index k_i of the multi-index $\boldsymbol{k} := (k_1, \ldots, k_d) \in \mathcal{I} \subset \mathbb{N}^d$ refers to the one-dimensional quadrature rule applied in the i'th dimension. The involved tensor product of quadrature rules is defined as

$$(Q_{k_1,1} \otimes \cdots \otimes Q_{k_d,1}) f := \sum_{i_{k_1}=1}^{n_{k_1,1}} \cdots \sum_{i_{k_d}=1}^{n_{k_d,1}} w_{i_{k_1},1} \cdots w_{i_{k_d},1} f(\eta_{i_{k_1},1}, \ldots, \eta_{i_{k_d},1}).$$

The multi-indices need to yield also an "increment by one" rule as in the one-dimensional case to fulfil the telescoping series. Consequently, for each valid multi-index $\boldsymbol{k}$ also each multi-index is valid which only differs from $\boldsymbol{k}$ by a decrement of exactly one index (as long as the decremented multi-index is still in $\mathbb{N}^d$):

$$\boldsymbol{k} \in \mathcal{I} \Rightarrow \tilde{\boldsymbol{k}} \in \mathcal{I} : \tilde{\boldsymbol{k}} := \boldsymbol{k} - \boldsymbol{e}_i \wedge k_i > 1, i \in \{1, \ldots, d\}. \tag{2.44}$$

$\boldsymbol{e}_i$ is the unit vector with the i'th component equal to 1. Eq. (2.43) together with the demand on the multi-indices in Eq. (2.44) defines the *general sparse grid construction* [71]. This notation contains the so-called *isotropic Smolyak algorithm* as well as the *anisotropic Smolyak algorithm*.

The (isotropic) Smolyak algorithm [71, 70, 169] is a sparse construction obtained by the set of multi-indices $\mathcal{I} := \{\boldsymbol{k} : |\boldsymbol{k}| \leq l + d - 1\}$ with $|\boldsymbol{k}| := \sum_{i=1}^{d} k_i$ and the given

order (level) l. The related convergence rate is described by $O(\frac{1}{n^r}(\log n)^{(d-1)(r+1)})$ for functions with bounded (mixed) partial derivatives up to an order of r [71]. Consequently, the Smolyak algorithm profits from the smoothness of the integrand and reaches even exponential convergence when $r \to \infty$. Furthermore, it is almost independent from the dimension except for a logarithmic factor. Surely in very high dimensions this factor becomes a problem. Either way the isotropic Smolyak algorithm considers all dimensions equally. That is suboptimal when the integrand contains different degrees of information in the particular dimensions.

The anisotropic Smolyak algorithm [71, 170] treats the dimensions with different importance. In this context, [170] a priori chooses the sequences of one-dimensional quadrature rules for each dimension differently. An adaptive algorithm to automatically construct an anisotropic sparse grid is presented in [71] and considered in more detail in the next passage. [107] introduces a modified Smolyak algorithm for functions with (blockwise) symmetric arguments. This strongly reduces the number of required function evaluations and even leads to dimension independence at least up to a certain order. [52] constructs the Smolyak algorithm for dimension $d := c{\cdot}b$ ($c \in \mathbb{N}$) from b-dimensional (randomised) QMC rules.

As already mentioned the authors of [71] propose an adaptive sparse grid construction to sample the integrand with respect to the corresponding importance of the particular dimensions. For this purpose the multi-indices are related to two disjoint sets, the *old* and the *active* ones. Initially, the old set is empty, the active set contains the first multi-index $\boldsymbol{k}_1 := (1, \ldots, 1)$. The corresponding error is computed by the intrinsic difference formula of the sparse grid construction. Then the multi-index is transferred to the old set. The increment of exactly one index of $\boldsymbol{k}_1$ leads to a new multi-index; this is done separately for each index resulting in a total of d new multi-indices. These multi-indices are — so to speak — the one-index-increment neighbours of $\boldsymbol{k}_1$. They are declared as active and their errors are computed as mentioned. The active multi-index with the highest error is assumed to indicate the dimensions in which a finer resolution is required. In the subsequent iteration this active multi-index $\boldsymbol{k}_{\epsilon_{max}}$ is transferred to the old set. Each one-index-increment neighbour of $\boldsymbol{k}_{\epsilon_{max}}$ is added to the active set for which all one-index-decrement neighbours are elements of the old set. Such iterations are performed either until a given estimated error is reached or a given time and effort is exceeded.

A nestedness of underlying one-dimensional quadrature rules is maintained by the Smolyak algorithm. In this thesis the isotropic Smolyak algorithm is applied with the Gauss-Legendre, Gauss-Hermite, Clenshaw-Curtis, or Kronrod-Patterson quadrature rule. A *delayed Kronrod-Patterson* quadrature rule [170] is also used. At this, the sequence of different levels for the Kronrod-Patterson quadrature rule is delayed by

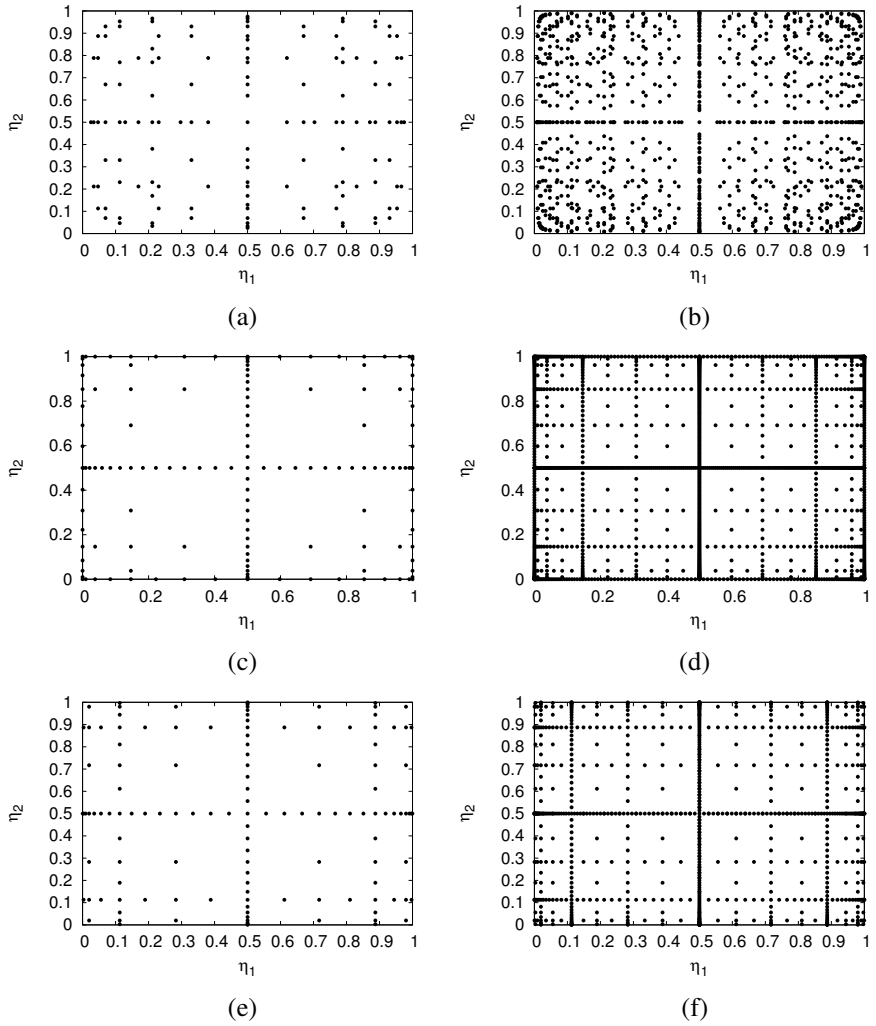
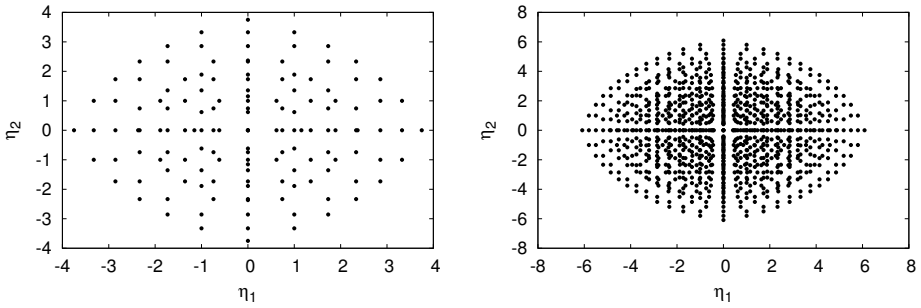


Figure 2.4: *Smolyak algorithm based on one-dimensional quadrature rules with weight function* $w(\eta) := 1$*: the figures show the constructed sparse grids regarding Gauss-Legendre (Figures (a) and (b) with* 137 *and* $1,009$ *sample-points), Clenshaw-Curtis (Figures (c) and (d) with* 145 *and* $1,537$ *sample-points), and Kronrod-Patterson (Figures (e) and (f) with* 129 *and* 829 *sample-points) quadrature rules.*



(a) (b)

Figure 2.5: *Smolyak algorithm based on Gauss-Hermite quadrature rule with Gaussian measure as the weight function: Figures (a) and (b) show the mentioned rule with* 139 *and* $1,012$ *sample-points.*

repeating levels. As a consequence the increase of the number of sample-points is reduced when incrementing level l. Figures 2.4 and 2.5 display some constructed sparse grids.

2.4 Projection of the Solution

A truncated PCE — $\sum_{\alpha\in\mathcal{I}^c} u_\alpha(\boldsymbol{x})\ \psi_\alpha(\boldsymbol{\theta}(\omega))$ — for the random field of a solution $u(\boldsymbol{x},\omega)$ can be obtained by determining each coefficient function (compare with Eq. (2.6))

$$u_\alpha(\boldsymbol{x}) = \frac{\mathbb{E}\left(u(\boldsymbol{x},\omega)\ \psi_\alpha(\boldsymbol{\theta}(\omega))\right)}{E(\psi_\alpha^2)} \tag{2.45}$$

through an integration [145, 123, 122] ($\mathbb{E}(\cdot) := \int_\Omega \cdot\, \mathrm{d}P(\omega)$ is the expectation operator). More precisely, while the denominator is usually analytically known, the enumerator is approximated by direct integration schemes (see Section 2.3). As a consequence the described projection scheme is non-intrusive.

In [121, 122] the scheme is applied to get a representation of fixed low rank which approximates the solution. There, new deterministic samples can be used — on the fly, so to speak — to improve the quality of the low-rank approximation.

2.5 Stochastic Collocation Methods

Stochastic collocation methods [11, 153, 66, 216, 20, 24, 26, 208, 62] approximate the PCE for the solution of a corresponding probabilistic model through model evaluations at a number of sample-points, the so-called *collocation points*. A general description for these methods is presented in the following. Two methods in particular are shortly outlined in Sections 2.5.1 and 2.5.2, namely a stochastic Lagrange interpolation [11, 154, 153, 66] and a least squares regression approach [20, 24, 25, 26]. A further scheme — not explicitly considered here — adaptively decomposes the sample space in subdomains, in which the solution is approximated through a stochastic collocation scheme [208, 63, 62].

A stochastic collocation method to approximate the solution u can be formulated by

$$\begin{aligned} u(\boldsymbol{x},\omega) \;&\approx\; \sum_{\alpha\in\mathcal{I}^c} u_\alpha(\boldsymbol{x})\psi_\alpha(\boldsymbol{\theta}(\omega)) \;=:\; u_c(\boldsymbol{x},\omega) \\ u_c(\boldsymbol{x},\omega_i) \;&\equiv\; u(\boldsymbol{x},\omega_i) \;+\; \epsilon_i \qquad\qquad \forall i \in \{1,\dots,n_p\}, \end{aligned} \tag{2.46}$$

where the first equation indicates the truncated PCE with N_S stochastic polynomials $\{\psi_\alpha\}_{\alpha\in\mathcal{I}^c}$ (defined on a random vector $\boldsymbol{\theta}$ of mutually independent random variables) and corresponding geometrical coefficient functions $\{u_\alpha\}_{\alpha\in\mathcal{I}^c}$. The second equation states that the approximation u_c needs to satisfy the exact solution u in each of the n_p collocation points $\boldsymbol{\theta}_i := \boldsymbol{\theta}(\omega_i)$ except for a small error $\epsilon_i \in \mathbb{R}$. The approximation u_c is an interpolation if $\epsilon_i = 0$ holds for all i.

To simplify matters a geometrical operator τ is applied to $u(\boldsymbol{x},\omega)$, restricting it to a random variable $u_\tau(\omega)$:

$$u_\tau(\omega) = \tau(u(\boldsymbol{x},\omega)). \tag{2.47}$$

τ may for example extract the solution at a special geometrical location or may integrate over (a part of) the geometrical domain. Then the stochastic collocation method applied onto the restricted solution can be written as

$$\begin{aligned} u_\tau(\omega) \;&\approx\; \sum_{\alpha\in\mathcal{I}^c} u_\alpha\psi_\alpha(\boldsymbol{\theta}(\omega)) \;=:\; u_{\tau_c}(\omega) \\ u_{\tau_c}(\omega_i) \;&\equiv\; u_\tau(\omega_i) \;+\; \epsilon_i \qquad\qquad \forall i \in \{1,\dots,n_p\}. \end{aligned} \tag{2.48}$$

This formulation can be expressed in a linear system

$$\underbrace{\begin{pmatrix} \psi_{\alpha_1}(\boldsymbol{\theta}_1) & \psi_{\alpha_2}(\boldsymbol{\theta}_1) & \cdots & \psi_{\alpha_{N_S}}(\boldsymbol{\theta}_1) \\ \vdots & \vdots & \ddots & \vdots \\ \psi_{\alpha_1}(\boldsymbol{\theta}_{n_p}) & \psi_{\alpha_2}(\boldsymbol{\theta}_{n_p}) & \cdots & \psi_{\alpha_{N_S}}(\boldsymbol{\theta}_{n_p}) \end{pmatrix}}_{=:\boldsymbol{\Psi}} \underbrace{\begin{pmatrix} u_{\alpha_1} \\ \vdots \\ u_{\alpha_{N_S}} \end{pmatrix}}_{=:\boldsymbol{u}} = \underbrace{\begin{pmatrix} u_{\tau_c,1} \\ \vdots \\ u_{\tau_c,n_p} \end{pmatrix}}_{=:\boldsymbol{u}_{\tau_c}}, \tag{2.49}$$

where $\boldsymbol{\Psi}$ is like a Vandermonde matrix and $u_{\tau_c,i} := u_{\tau_c}(\omega_i)$. The system is under-determined, determined, or over-determined if $n_p < N_S$, $n_p = N_S$, or $n_p > N_S$.

When the stochastic collocation method is applied to the non-restricted solution $u(\boldsymbol{x}, \omega)$ in Eq. (2.46), the coefficient functions $\{u_\alpha\}_{\alpha \in \mathcal{I}^c}$ need to be discretised in the geometrical space. For this purpose linear finite elements are applied here leading to N_X geometrical DoFs and corresponding linear basis functions. For each geometrical DoF a linear system like in Eq. (2.49) can be formulated separately: $\boldsymbol{\Psi}\boldsymbol{u}^{(k)} = \boldsymbol{u}_{\tau_c}^{(k)}$ is the linear system corresponding to the k'th geometrical DoF. Then the entire system to be solved can be written as a block diagonal system:

$$\begin{pmatrix} \boldsymbol{\Psi} & 0 & \cdots & 0 \\ 0 & \boldsymbol{\Psi} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \boldsymbol{\Psi} \end{pmatrix} \begin{pmatrix} \boldsymbol{u}^{(1)} \\ \boldsymbol{u}^{(2)} \\ \vdots \\ \boldsymbol{u}^{(N_X)} \end{pmatrix} = \begin{pmatrix} \boldsymbol{u}_{\tau_c}^{(1)} \\ \boldsymbol{u}_{\tau_c}^{(2)} \\ \vdots \\ \boldsymbol{u}_{\tau_c}^{(N_X)} \end{pmatrix}. \tag{2.50}$$

The choice of stochastic polynomials influences the condition number of the coefficient matrix in Eq. (2.49). Thus it may become an additional problem to find useful polynomials. Two different kinds of polynomials can be considered: polynomials whose orders are not interlocked with the number of collocation points, and polynomials whose orders are interlocked. Non-interlocked basis polynomials are for instance multi-dimensional Hermite polynomials. (In the latter case reasonably $\boldsymbol{\theta}$ is a Gaussian vector.) Then the collocation points may be obtained through MC, QMC, or Smolyak sampling techniques. Interlocked basis polynomials are for instance given in the Lagrange interpolation discussed in the next section.

A further detail in all mentioned collocation schemes is that higher order basis polynomials may lead to oscillatory behaviour especially at the boundary of the underlying space — a problem which is more difficult to control in the context of interlocked basis polynomials, as the polynomial order is linked to the number of collocation points. A regression approach on the basis of an over-determined system may be

used to counter the undesirable oscillatory effect. A stochastic Lagrange interpolation and its sparse construction are addressed in Section 2.5.1. A regression approach for an over-determined system is presented in Section 2.5.2.

2.5.1 Stochastic Lagrange Interpolation

A stochastic Lagrange interpolation is applied in [11, 154, 153, 66, 63, 62]. It is an interlocked interpolation. Its basis polynomials are directly constructed from a set of collocation points $\{\boldsymbol{\theta}_i\}_{i\in\{1,\dots,n_p\}}$. At this, a Lagrange basis polynomial corresponds to a collocation point so that consequently $n_p = N_S$ holds. A particularity of Lagrange collocation is that the coefficient of a Lagrange basis polynomial is directly the response $u_\tau(\omega_i)$ of the corresponding collocation point $\boldsymbol{\theta}(\omega_i)$. Accordingly, no linear system needs to be solved, as the coefficient matrix $\boldsymbol{\Psi}$ of Eq. (2.49) is simply the identity matrix: $\boldsymbol{\Psi} := \boldsymbol{I}$. This is a strength. However, the number of collocation points is equal to the order of a Lagrange polynomial. Therefore the Lagrange interpolation is prone to the mentioned undesirable oscillatory behaviour.

A full tensor construction inside the Lagrange interpolation [11] means that the set of points to construct each Lagrange polynomial is the set of all collocation points. That underlies the curse of dimension, and consequently the scheme is not applicable for higher dimensions. For higher dimensions the Lagrange polynomials can be constructed in a sparse manner. That is denoted as the *stochastic sparse grid collocation* [154, 153, 66]. However, the corresponding collocation scheme does not anymore fulfil the mentioned characteristic of an interpolation to satisfy the exact response at the collocation points. The sparse construction is exactly a Smolyak one (see also Section 2.3.2).

The isotropic Smolyak construction — in which each dimension is considered with the same attention — is especially presented in [154]. For (strongly) anisotropic solutions this is not optimal, as collocation points could be put aside in dimensions of less information content (while additional sample-points could be added in dimensions of more information content). For such cases the Smolyak construction is performed in an anisotropic manner in [66, 153], in which a corresponding adaptive algorithm is presented as well (this algorithm was introduced in [71] for numerical integration and is shortly outlined in Section 2.3.2.2).

A stochastic Lagrange interpolation scheme is used in [63, 62] to approximate the solution in the subdomains of an adaptively decomposed sample space. A comparable scheme which is constructed from sparse tensor products of one-dimensional basis

functions and supports an adaptive refinement of the domain discretisation and the polynomial order is presented in [36, 35]. It is not explicitly introduced for stochastic purposes but implicitly covers that topic.

2.5.2 Least Squares Regression Approach

The least squares regression approach in [20, 24, 25, 26] tries to determine the coefficients $\{u_\alpha\}_{\alpha\in\mathcal{I}^c}$ of Eq. (2.48) by solving the normal equation $\boldsymbol{\Psi}^T\boldsymbol{\Psi}\,\boldsymbol{u} = \boldsymbol{\Psi}^T\,\boldsymbol{u}_\tau$ of Eq. (2.49) with an over-determined system ($n_p > N_S$) and $\boldsymbol{u}_\tau := (u_\tau(\omega_1),\ldots,u_\tau(\omega_{n_p}))^T$ instead of $\boldsymbol{u}_{\tau_c}$. Algorithms to adaptively construct a sparse PCE through such a regression approach are presented in [24, 25, 26] and shortly considered in Section 2.1.1.4.

The normal equation may be solved by a QR decomposition [74]. However, when $\boldsymbol{\Psi}^T\boldsymbol{\Psi}$ is rank deficient, the QR decomposition with column pivoting, or a singular value decomposition (SVD) may be used [74]. The latter is applied here and shortly outlined in the following. The SVD of $\boldsymbol{\Psi}$ is $\boldsymbol{\Psi} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$ with a diagonal matrix $\boldsymbol{\Sigma}$ — storing the singular values — and orthogonal matrices $\boldsymbol{U}$ and $\boldsymbol{V}$. Substituting the SVD in the normal equation and rearranging the resulting equation yields $\boldsymbol{\Sigma}^2\hat{\boldsymbol{u}} = \boldsymbol{b}$ with $\hat{\boldsymbol{u}} := \boldsymbol{V}^T\boldsymbol{u}$ and $\boldsymbol{b} := \boldsymbol{V}^T\boldsymbol{\Psi}^T\boldsymbol{u}_\tau$. When a possible rank deficiency is taken into account, the formula ends in

$$\hat{u}_i = \begin{cases} \frac{b_i}{s_i^2} & \text{for } s_i \neq 0 \\ 0 & \text{else} \end{cases} \qquad \boldsymbol{u} = \boldsymbol{V}\hat{\boldsymbol{u}} \tag{2.51}$$

with the i'th component b_i of vector $\boldsymbol{b}$, the i'th component $\hat{u}_i$ of vector $\hat{\boldsymbol{u}}$, and the i'th singular value s_i.

2.6 Stochastic Galerkin Method (SGM)

The stochastic Galerkin method (SGM) [73, 129, 128, 12, 217, 127, 3, 57, 130] discretises the weak formulation of a stochastic partial differential equation in the stochastic space to obtain an approximation for the solution.

The weak formulation — or shorter: the weak form — is demonstratively derived and discretised in Section 2.6.1 for the stochastic stationary groundwater flow problem (see Eq. (2.34) ff.). The non-Gaussian hydraulic conductivity is described once by a truncated PCE and once by a truncated KLE, and the resulting linear systems to be solved are obtained in Section 2.6.2. Some annotations about how the linear systems may be solved are outlined in Section 2.6.3.

2.6.1 The Weak Formulation and the Discretisation of the Solution

The stochastic stationary groundwater flow problem in Eq. (2.34) ff. is rearranged in its residual representation, weighted by a test function v, and integrated over the geometric and stochastic spaces, so that the problem can be reformulated by

$$\mathbb{E}\left(\int_X \Big(\nabla_{\boldsymbol{x}} \cdot (\kappa(\boldsymbol{x},\omega)\nabla_{\boldsymbol{x}} u(\boldsymbol{x},\omega)) \; + \; f(\boldsymbol{x},\omega) \Big)\, v(\boldsymbol{x},\omega)\, \mathrm{d}\boldsymbol{x}\right) = 0 \qquad \forall v \in V \tag{2.52}$$

with $u \in V := H_0^2(X) \otimes L_p^2(\Omega)$ and $\mathbb{E}(\cdot) := \int_\Omega \cdot \, \mathrm{d}P(\omega)$. To simplify matters, the essential boundary conditions are assumed to be zero. The weak formulation $a(u,v) = l(v)$ with the bilinear form $a(u,v)$ and the linear form $l(v)$ is obtained by a partial integration and the utilisation of the divergence theorem [181]. It can be written as

$$\begin{aligned} \mathbb{E}\left(\int_X \kappa(\boldsymbol{x},\omega)\nabla_{\boldsymbol{x}} u(\boldsymbol{x},\omega)\nabla_{\boldsymbol{x}} v(\boldsymbol{x},\omega)\, \mathrm{d}\boldsymbol{x}\right) \; = \\ \mathbb{E}\left(\int_X f(\boldsymbol{x},\omega) v(\boldsymbol{x},\omega)\, \mathrm{d}\boldsymbol{x}\right) \; + \; \mathbb{E}\left(\int_{\partial X_N} t(\boldsymbol{x},\omega) v(\boldsymbol{x},\omega)\, \mathrm{d}\boldsymbol{x}\right) \qquad \forall v \in V' \end{aligned} \tag{2.53}$$

with $u \in V' := H_0^1(D) \otimes L_p^2(\Omega)$ [129, 128]. The symmetric bilinear form has to be positive-definite. The divergence theorem introduces the relation of the flux t at the boundary ∂X_N to the solution in the geometrical domain. As the name "weak form" stipulates, the solution u has less requirements, as it only needs to be differentiable once.

The solution u is now discretised by finitely many ansatz functions (basis functions) $\{\zeta_k\}_{k\in 1,\ldots,N}$ to obtain an approximation for u: $u(\boldsymbol{x},\omega) \approx u_h(\boldsymbol{x},\omega) := \sum_{k=1}^{N} u_k \zeta_k(\boldsymbol{x},\omega)$. In the context of the SGM the ansatz functions are also used

for the discretisation of the test functions. In this manner the weak form in Eq. (2.53) is transferred into a finite dimensional space $V'_N \subset V'$:

$$\begin{aligned} &\mathbb{E}\left(\int_X \kappa(\boldsymbol{x},\omega)\Big(\sum_{k=1}^{N} u_k \nabla_{\boldsymbol{x}}\zeta_k(\boldsymbol{x},\omega)\Big)\nabla_{\boldsymbol{x}}\zeta_l(\boldsymbol{x},\omega)\,\mathrm{d}\boldsymbol{x}\right) = \\ &\mathbb{E}\left(\int_X f(\boldsymbol{x},\omega)\zeta_l(\boldsymbol{x},\omega)\,\mathrm{d}\boldsymbol{x}\right) + \mathbb{E}\left(\int_{\partial X_N} t(\boldsymbol{x},\omega)\zeta_l(\boldsymbol{x},\omega)\,\mathrm{d}\boldsymbol{x}\right) \\ &\qquad\qquad \forall l \in \{1,\dots,N\}. \end{aligned} \tag{2.54}$$

The ansatz functions — and consequently the test functions — can be represented by the tensor product of ansatz functions separately for the geometrical and the stochastic spaces: $\zeta_k(\boldsymbol{x},\omega) = \phi_i(\boldsymbol{x}) \cdot \psi_\alpha(\boldsymbol{\theta}(\omega))$ with $i \in \{1,\dots,N_X\}$, $\alpha \in \mathcal{I}^c$, $N_S := |\mathcal{I}^c|$ and $N = N_X \cdot N_S$. Accordingly, Eq. (2.54) becomes

$$\begin{aligned} &\sum_{i=1}^{N_X}\sum_{\alpha\in\mathcal{I}^c} u_{i,\alpha}\mathbb{E}\Big(\psi_\alpha(\boldsymbol{\theta}(\omega))\psi_\beta(\boldsymbol{\theta}(\omega))\int_X \nabla_{\boldsymbol{x}}\phi_i(\boldsymbol{x})\kappa(\boldsymbol{x},\omega)\nabla_{\boldsymbol{x}}\phi_j(\boldsymbol{x})\mathrm{d}\boldsymbol{x}\Big) = \\ &\mathbb{E}\left(\psi_\beta(\boldsymbol{\theta}(\omega))\Big(\int_X f(\boldsymbol{x},\omega)\phi_j(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} + \int_{\partial X_N} t(\boldsymbol{x},\omega)\phi_j(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}\Big)\right) \\ &\qquad\qquad \forall\, j \in \{1,\dots,N_X\},\ \beta \in \mathcal{I}^c. \end{aligned} \tag{2.55}$$

The hydraulic conductivity is discretised in the next section.

2.6.2 Discretisation of the Material Parameter Field

A discretisation of the hydraulic conductivity κ has not been considered until now. Because the hydraulic conductivity is a positive material parameter it is represented by a non-Gaussian random field [129, 98]. For the sake of simplicity, homogeneous essential boundary conditions, zero natural boundary conditions, and a deterministic source/sink term are assumed. This configuration simplifies Eq. (2.55) to

$$\begin{aligned} &\sum_{i=1}^{N_X}\sum_{\alpha\in\mathcal{I}^c} u_{i,\alpha}\mathbb{E}\Big(\psi_\alpha(\boldsymbol{\theta}(\omega))\psi_\beta(\boldsymbol{\theta}(\omega))\int_X \nabla_{\boldsymbol{x}}\phi_i(\boldsymbol{x})\kappa(\boldsymbol{x},\omega)\nabla_{\boldsymbol{x}}\phi_j(\boldsymbol{x})\mathrm{d}\boldsymbol{x}\Big) = \\ &\mathbb{E}\left(\psi_\beta(\boldsymbol{\theta}(\omega))\int_X f(\boldsymbol{x})\phi_j(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}\right) \qquad \forall\, j \in \{1,\dots,N_X\},\ \beta \in \mathcal{I}^c. \end{aligned} \tag{2.56}$$

κ is discretised by a truncated PCE in the first paragraph and by a truncated KLE in the second paragraph. In both cases the resulting linear systems are derived. The

linear system corresponding to a KL discretisation of the hydraulic conductivity is computationally more practical. Both discretisations are considered in the numerical experiments in Section 4.1.

The Discretisation of κ by a Truncated PCE The truncated PCE κ_{c_γ} for the hydraulic conductivity κ is represented by (see Section 2.1.1.1)

$$\kappa(\boldsymbol{x},\omega) \approx \kappa_{c_\gamma}(\boldsymbol{x},\omega) := \sum_{\gamma\in\mathcal{I}^{c_\gamma}} \kappa_\gamma(\boldsymbol{x})\,\psi_\gamma(\boldsymbol{\theta}(\omega)) \tag{2.57}$$

with the set of stochastic polynomials indexed by $\mathcal{I}^{c_\gamma}$. A substitution of κ by κ_{c_γ} in Eq. (2.56) [129] leads to

$$\begin{aligned}\sum_{\gamma\in\mathcal{I}^{c_\gamma}} \sum_{i=1}^{N_X} \sum_{\alpha\in\mathcal{I}^c} u_{i,\alpha}\mathbb{E}\left(\psi_\alpha(\boldsymbol{\theta}(\omega))\psi_\beta(\boldsymbol{\theta}(\omega))\psi_\gamma(\boldsymbol{\theta}(\omega))\right)\cdot \\ \int_X \nabla_{\boldsymbol{x}}\phi_i(\boldsymbol{x})\kappa_\gamma(\boldsymbol{x})\nabla_{\boldsymbol{x}}\phi_j(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} = \\ \mathbb{E}\left(\psi_\beta(\boldsymbol{\theta}(\omega))\int_X f(\boldsymbol{x})\phi_j(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}\right) \qquad \forall\, j\in\{1,\dots,N_X\},\ \beta\in\mathcal{I}^c.\end{aligned} \tag{2.58}$$

The integral terms on the left-hand side associated with the geometrical space can be summarised by the stiffness matrix $\boldsymbol{K}_\gamma$ with $[\boldsymbol{K}_\gamma]_{i,j} := \int_X \nabla_{\boldsymbol{x}}\phi_i(\boldsymbol{x})\kappa_\gamma(\boldsymbol{x})\nabla_{\boldsymbol{x}}\phi_j(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x}$ (the stiffness matrix may be obtained by a simulation code based on deterministic finite elements). Then the coefficients $\{u_{i,\alpha}\}_{i\in\{1,\dots,N_X\}}$ may be composed to a vector $\boldsymbol{u}_\alpha := (u_{1,\alpha},\dots,u_{N_X,\alpha})^T$ for all $\alpha\in\mathcal{I}^c$ to express the sum in Eq. (2.58) associated with the geometrical discretisation. The geometrical coefficients of the right-hand side may also be composed to a vector $\boldsymbol{f}_X := (\int_X f(\boldsymbol{x})\phi_1\,\mathrm{d}\boldsymbol{x},\dots,\int_X f(\boldsymbol{x})\phi_{N_X}\,\mathrm{d}\boldsymbol{x})^T$. Consequently, Eq. (2.58) becomes

$$\begin{aligned}\sum_{\gamma\in\mathcal{I}^{c_\gamma}} \sum_{\alpha\in\mathcal{I}^c} \mathbb{E}\left(\psi_\alpha(\boldsymbol{\theta}(\omega))\psi_\beta(\boldsymbol{\theta}(\omega))\psi_\gamma(\boldsymbol{\theta}(\omega))\right)\ \boldsymbol{K}_\gamma\boldsymbol{u}_\alpha \\ = \mathbb{E}\left(\psi_\beta(\boldsymbol{\theta}(\omega))\right)\boldsymbol{f}_X \quad \forall\,\beta\in\mathcal{I}^c.\end{aligned} \tag{2.59}$$

Analogously, the sum which results from the stochastic discretisation of the solution may be written in a matrix notation. Then the integral terms on the left-hand side (associated with the stochastic space) are summarised by a matrix $\boldsymbol{\Delta}_\gamma$ with $[\boldsymbol{\Delta}_\gamma]_{\alpha,\beta} = \mathbb{E}\left(\psi_\alpha(\boldsymbol{\theta}(\omega))\psi_\beta(\boldsymbol{\theta}(\omega))\psi_\gamma(\boldsymbol{\theta}(\omega))\right)$. The coefficients of the solution are

described by vector $\boldsymbol{u} := (\boldsymbol{u}_{\alpha_1}^T, \ldots, \boldsymbol{u}_{\alpha_{N_S}}^T)^T$. The right-hand side can be summarised by vector $\boldsymbol{f} = (\boldsymbol{f}_{\beta_1}^T, \ldots, \boldsymbol{f}_{\beta_{N_S}}^T)^T$ with $\boldsymbol{f}_\beta := \mathbb{E}\left(\psi_\beta(\boldsymbol{\theta}(\omega))\boldsymbol{f}_X\right)$. The tensor product between the geometric and the stochastic spaces is expressed by the Kronecker product $\otimes$ [85] for matrices. Consequently, Eq. (2.59) becomes a linear system [129]

$$\left(\sum_\gamma \boldsymbol{\Delta}_\gamma \otimes \boldsymbol{K}_\gamma\right)\boldsymbol{u} = \boldsymbol{f}. \tag{2.60}$$

In practice the Kronecker product is usually not performed, as the resulting coefficient matrix may become too large. Therefore, another notation [85] for the Kronecker product is preferred for numerical schemes:

$$\sum_\gamma \boldsymbol{K}_\gamma \boldsymbol{U} \boldsymbol{\Delta}_\gamma = \boldsymbol{F} \tag{2.61}$$

with $\boldsymbol{U} := [\boldsymbol{u}_{\alpha_1}, \ldots, \boldsymbol{u}_{\alpha_{N_S}}] \in \mathbb{R}^{N_X \times N_S}$, $\boldsymbol{F} := [\boldsymbol{f}_{\beta_1}, \ldots, \boldsymbol{f}_{\beta_{N_S}}] \in \mathbb{R}^{N_X \times N_S}$.

The Discretisation of κ by a Truncated KLE The discretisation of the hydraulic conductivity κ by a truncated KLE κ_l (see Section 2.1.1.2) leads to a linear system, which is computationally more practical with respect to both memory and performance demands [129, 98]. κ_l is represented by

$$\kappa(\boldsymbol{x}, \omega) \;\approx\; \kappa_l(\boldsymbol{x}, \omega) \;:=\; \sum_{i=0}^{l} \sqrt{\lambda_i}\, \kappa_i(\boldsymbol{x}) \sum_{\gamma \in \mathcal{I}^{c_\gamma}} \xi_{i,\gamma}\, \psi_\gamma(\boldsymbol{\theta}(\omega)). \tag{2.62}$$

In contrast to Section 2.1.1.2, the expectation $\bar{\kappa}$ is implicated in the sum, more precisely $\kappa_0(\boldsymbol{x}) := \bar{\kappa}(\boldsymbol{x})$, $\lambda_0 := 1$, as well as $\xi_{0,\mathbf{0}} = 1$ and $\xi_{0,\gamma} = 0$ for $\gamma \neq \mathbf{0}$ hold.

κ_l now substitutes κ in Eq. (2.56). The derivation of the linear system in matrix notation is abbreviated here, as it is analogous to the one in the previous paragraph. The linear system can be written as [129]

$$\left(\sum_{i=0}^{l} \sum_{\gamma \in \mathcal{I}^c} \sqrt{\lambda_i}\, \xi_{i,\gamma}\, \boldsymbol{\Delta}_\gamma \otimes \boldsymbol{K}_i\right)\boldsymbol{u} = \boldsymbol{f}$$

$$\left(\sum_{i=0}^{l} \boldsymbol{\Delta}_i \otimes \boldsymbol{K}_i\right)\boldsymbol{u} = \boldsymbol{f} \tag{2.63}$$

$$\sum_{i=0}^{l} \boldsymbol{K}_i\, \boldsymbol{U}\, \boldsymbol{\Delta}_i = \boldsymbol{F} \tag{2.64}$$

with $\boldsymbol{\Delta}_i := \sum_{\gamma \in \mathcal{I}^c} \sqrt{\lambda_i}\, \xi_{i,\gamma}\, \boldsymbol{\Delta}_\gamma$. The vectors $\boldsymbol{u}$ and $\boldsymbol{f}$ as well as the matrices $\boldsymbol{\Delta}_\gamma$, $\boldsymbol{U}$, and $\boldsymbol{F}$ are defined as in the previous paragraph. The stiffness matrix $\boldsymbol{K}_i$ is defined by $[\boldsymbol{K}_i]_{j,k} := \int_X \nabla_{\boldsymbol{x}} \phi_j(\boldsymbol{x}) \kappa_i(\boldsymbol{x}) \nabla_{\boldsymbol{x}} \phi_k(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x}$ with eigenfunction κ_i (when $i > 0$) or expectation κ_0 (when $i = 0$).

2.6.3 Solving the Linear System

The discretisation of the stochastic stationary groundwater flow problem in Eq. (2.34) ff. by the stochastic Galerkin method leads to a linear system to be solved. The linear system is presented in Eqs. (2.60) and (2.61) — where the hydraulic conductivity is discretised by a PCE — or in Eq. (2.64) — where the hydraulic conductivity is discretised by a KLE. Each linear system is positive-definite (when enough terms are included) and symmetric [129, 98].

The linear system may be solved in different ways [129, 205, 183, 98, 130, 13, 103, 158, 111]. It may be solved directly, but the system is usually too large. Efficient iterative solvers may be used like the preconditioned conjugate gradient method [184], which searches for the minimum of the residual in conjugate directions. The involved preconditioner establishes a reduction of the condition number to improve the convergence. In [129] a block-Jacobi preconditioner is applied. In contrast, a preconditioner which uses information of the entire coefficient matrix is proposed in [205] and displays a better convergence. A huge number of degrees of freedom may require even more sophisticated iterative schemes like the multilevel methods [30, 203]. These schemes operate on discretisations of different granularity to reduce the low-frequency error on coarse grained discretisations and the high-frequency error on fine grained discretisations. This is applied in [98] for the discretisation of the stochastic space and in [183] for the discretisation of the geometrical space.

Other approaches search for a low-rank approximation of the solution [130, 13, 103, 158, 111]. In [130, 13, 103] the low-rank representation is embedded in an iterative solver; in each iteration a rank increase happens, which is then handled through a rank truncation algorithm. In [158, 111] successive rank updates are proposed which minimise the expectation of the total potential energy. A solution space adaption is additionally performed in our technical report [111] and presented in the next section.

All of these different schemes to solve the SGM-discretised problem are summarised under the term “SGM-based schemes”. This term is especially used in Chapter 3.

2.7 A Successive Rank-One Update and a Solution Space Adaption in the SGM

This section focuses on successive rank update schemes [158, 111] to approximate a low-rank representation for the solution of a stochastic partial differential equation discretised by the SGM. A rank update is obtained under the proposition to minimise the expectation of the total potential energy of the problem. However, an approximation is suboptimal when it is achieved by successive rank updates only [158]. Therefore, a global update of a low-rank approximation may be performed. In [158] a more general approach is actually presented namely for a successive rank-r update or an approximation of a fixed rank — that includes of course a successive rank-one update. It is published under the term *generalized spectral decomposition* (GSD). In our technical report [111] we construct a successive rank-one update scheme (it can be identified as a special configuration of the GSD). Further, an optimisation scheme for a rank-r approximation is developed in [111], which is based on the algorithm of the proposed rank-one update. Additionally, an adaptive construction of the solution space is presented in [111], which is comparable to the idea in [143] (and was simultaneously developed). The approaches in [111] are a contribution of this thesis and summarised here under the term *variational low-rank approach with successive rank-one updates*, abbreviated by VLR-SR1U.

The successive rank update schemes provide a kind of a greedy approximation [200], because a next rank update tries to extract most of the remaining energy. The convergence of greedy rank-one update schemes is recently proved for linear systems of full rank in [7].

The GSD and the VLR-SR1U scheme are presented for the stochastic stationary groundwater flow problem (see Eq. (2.34) ff). Thus, the starting point of both schemes is the SGM-discretised problem in form of the linear system (compare Eq. (2.64))

$$\sum_{i=0}^{l} \boldsymbol{K}_i \, \boldsymbol{U} \, \boldsymbol{\Delta}_i \;=\; \boldsymbol{F} \qquad \Longleftrightarrow \qquad \mathcal{A}(\boldsymbol{U}) \;=\; \boldsymbol{F}. \tag{2.65}$$

This problem is now reformulated in a minimisation problem of its total potential energy (the expectation is intrinsically involved in that system), which is also identified as the variational formulation in the literature:

$$\mathcal{E}(\boldsymbol{U}) \;:=\; \frac{1}{2}\mathcal{A}(\boldsymbol{U}) \;:\; \boldsymbol{U} \;-\; \boldsymbol{F} \;:\; \boldsymbol{U} \;\longrightarrow\; \min\,. \tag{2.66}$$

The involved inner product $A : B := \sum_{i,j} A_{ij} B_{ij}$ with $A, B \in \mathbb{R}^{n_1 \times n_2}$ is the so-called *Frobenius* product.

The GSD from [158] is addressed in Section 2.7.1. The basic VLR-SR1U scheme, the hereupon based optimisation scheme and the adaptive construction of the solution space to obtain a sparse PCE of low rank are discussed in Section 2.7.2.

2.7.1 The Generalized Spectral Decomposition

The motivation of [158, 157, 156] is to find a spectral decomposition for the solution of a stochastic partial differential equation. The scheme proposed in these publications is based on a minimisation of the expectation of the total potential energy principle, referred to as the generalized spectral decomposition (GSD). It is described in the following.

A low-rank ansatz $\boldsymbol{U} = \boldsymbol{G}\boldsymbol{H}^T$ (see Eq. (2.31)) is chosen for the solution in Eq. (2.66). Then, by using the trace operator and its properties the operator $\mathcal{E}$ can be rewritten as

$$\begin{aligned} \mathcal{E}(\boldsymbol{G}\boldsymbol{H}^T) &= \frac{1}{2}\mathcal{A}(\boldsymbol{G}\boldsymbol{H}^T) : \boldsymbol{G}\boldsymbol{H}^T - \boldsymbol{F} : \boldsymbol{G}\boldsymbol{H}^T \\ &= \operatorname{tr}\left(\frac{1}{2} \sum_{i=0}^{l} \boldsymbol{G}^T \boldsymbol{K}_i \boldsymbol{G}\boldsymbol{H}^T \boldsymbol{\Delta}_i \boldsymbol{H} - \boldsymbol{G}^T \boldsymbol{F}\boldsymbol{H} \right). \end{aligned}$$

To follow the minimisation problem means on the one hand to differentiate $\mathcal{E}(\boldsymbol{G}\boldsymbol{H}^T)$ with respect to $\boldsymbol{G}$ and on the other hand with respect to $\boldsymbol{H}$ ($\delta\boldsymbol{G}$ and $\delta\boldsymbol{H}$ are the appropriate variations):

$$\begin{aligned} \frac{\partial \mathcal{E}(\boldsymbol{G}\boldsymbol{H}^T)}{\partial \boldsymbol{G}}(\delta\boldsymbol{G}) &= \operatorname{tr}\left(\sum_{i=0}^{l} \delta\boldsymbol{G}^T \boldsymbol{K}_i \boldsymbol{G}\boldsymbol{H}^T \boldsymbol{\Delta}_i \boldsymbol{H} - \delta\boldsymbol{G}^T \boldsymbol{F}\boldsymbol{H} \right) \\ &= \sum_{i=0}^{l} \boldsymbol{K}_i \boldsymbol{G}\boldsymbol{H}^T \boldsymbol{\Delta}_i \boldsymbol{H} : \delta\boldsymbol{G} - \boldsymbol{F}\boldsymbol{H} : \delta\boldsymbol{G} \\ \frac{\partial \mathcal{E}(\boldsymbol{G}\boldsymbol{H}^T)}{\partial \boldsymbol{H}}(\delta\boldsymbol{H}) &= \operatorname{tr}\left(\sum_{i=0}^{l} \delta\boldsymbol{H}^T \boldsymbol{\Delta}_i \boldsymbol{H}\boldsymbol{G}^T \boldsymbol{K}_i \boldsymbol{G} - \delta\boldsymbol{H}^T \boldsymbol{F}^T \boldsymbol{G} \right) \\ &= \sum_{i=0}^{l} \boldsymbol{\Delta}_i \boldsymbol{H}\boldsymbol{G}^T \boldsymbol{K}_i \boldsymbol{G} : \delta\boldsymbol{H} - \boldsymbol{F}^T \boldsymbol{G} : \delta\boldsymbol{H}. \end{aligned}$$

To find its root respectively implies $\frac{\partial \mathcal{E}(\boldsymbol{G}\boldsymbol{H}^T)}{\partial \boldsymbol{G}}(\delta \boldsymbol{G}) \equiv 0$ and $\frac{\partial \mathcal{E}(\boldsymbol{G}\boldsymbol{H}^T)}{\partial \boldsymbol{H}}(\delta \boldsymbol{H}) \equiv 0$. The denoted orthogonality to each variation $\delta \boldsymbol{G}$ and analogously $\delta \boldsymbol{H}$ identifies the left-hand side of each inner product to be the zero matrix. As a consequence, the following coupled system has to be solved, the first equation to get $\boldsymbol{G}$ and the second one to get $\boldsymbol{H}$:

$$
\begin{aligned}
&\mathcal{G}(\boldsymbol{G}) \;=\; \mathcal{F}(\boldsymbol{F}) \\
&\qquad \text{with} \quad \mathcal{G}(\boldsymbol{G}) \;:=\; \sum_{i=0}^{l} \boldsymbol{K}_i\, \boldsymbol{G}\boldsymbol{H}^T\, \boldsymbol{\Delta}_i \boldsymbol{H}, \quad \mathcal{F}(\boldsymbol{F}) := \boldsymbol{F}\boldsymbol{H}, \\
&\mathcal{H}(\boldsymbol{H}) \;=\; \bar{\mathcal{F}}(\boldsymbol{F}) \\
&\qquad \text{with} \quad \mathcal{H}(\boldsymbol{H}) := \sum_{i=0}^{l} \boldsymbol{\Delta}_i\, \boldsymbol{H}\boldsymbol{G}^T\, \boldsymbol{K}_i \boldsymbol{G}, \quad \bar{\mathcal{F}}(\boldsymbol{F}) := \boldsymbol{F}^T \boldsymbol{G}.
\end{aligned}
\tag{2.67}
$$

The coupled system is expressed in form of operators acting on the particular unknown for the left-hand sides and on $\boldsymbol{F}$ for the right-hand sides. This notation is introduced to simplify subsequent algorithmic descriptions. Another notation clarifies the coupling. To do this the functions $\tau_{\boldsymbol{G}} : \boldsymbol{H} \mapsto \boldsymbol{G}$ and $\tau_{\boldsymbol{H}} : \boldsymbol{G} \mapsto \boldsymbol{H}$ are introduced [157]. Then the coupling may be formulated as $\tau := \tau_{\boldsymbol{G}} \circ \tau_{\boldsymbol{H}}$ or $\bar{\tau} := \tau_{\boldsymbol{H}} \circ \tau_{\boldsymbol{G}}$, so that $\boldsymbol{G} = \tau(\boldsymbol{G}) = \tau_{\boldsymbol{G}}(\tau_{\boldsymbol{H}}(\boldsymbol{G}))$ or $\boldsymbol{H} = \bar{\tau}(\boldsymbol{H}) = \tau_{\boldsymbol{H}}(\tau_{\boldsymbol{G}}(\boldsymbol{H}))$.

In [157] different numerical iterative schemes are presented to solve the coupled system in Eq. (2.67): a subspace iteration algorithm (*SI-GSD*), an Arnoldi type algorithm (*A-GSD*), and a restarting algorithm. SI-GSD computes a representation $\boldsymbol{G}\boldsymbol{H}^T$ of fixed rank r by solving Eq. (2.67) alternatingly in an iterative process; at this, the updated geometrical part $\boldsymbol{G}$ is orthonormalised in the Euclidean space in each iteration step. The resulting low-rank representation is optimal in the sense of minimising the expectation of the system's energy in comparison with all other representations of the same rank.

Finding the optimal representation of rank r is equivalent to solving an eigenproblem in which $\boldsymbol{G}$ contains the most significant eigenvectors. In this regard, A-GSD [157] tries to find these r eigenvectors and then computes $\boldsymbol{H}$ in the same manner as SI-GSD. It may happen that the r eigenvectors are not the most relevant ones, so that the resulting low-rank representation becomes only an approximation of that optimal one from the SI-GSD. However, the A-GSD has computational advantages compared to the SI-GSD. The quality of the A-GSD output can be increased by computing more than r eigenvectors and then choosing the r most significant ones. For both schemes rank r is predefined. When the resulting rank-r representation is of insufficient accuracy of course the schemes can be applied again for a higher rank, but

that means to recompute the whole low-rank representation. This can be inefficient, so an efficient alternative is discussed next.

An existing low-rank representation $\boldsymbol{G}_r\boldsymbol{H}_r^T$ of rank r may be updated to higher rank by another low-rank representation $\hat{\boldsymbol{G}}_{\hat{r}}\hat{\boldsymbol{H}}_{\hat{r}}^T$ of rank $\hat{r}$, so that the simple concatenation $[\boldsymbol{G}_r, \hat{\boldsymbol{G}}_{\hat{r}}][\boldsymbol{H}_r, \hat{\boldsymbol{H}}_{\hat{r}}]^T =: \boldsymbol{G}_{r+\hat{r}}\boldsymbol{H}_{r+\hat{r}}^T$ delivers a representation of rank not more than $r + \hat{r}$. $\hat{\boldsymbol{G}}_{\hat{r}}\hat{\boldsymbol{H}}_{\hat{r}}^T$ is computed in [157] by solving the coupled system in Eq. (2.67), in which $\boldsymbol{F}$ (the original right-hand side) is replaced by the residual induced by $\boldsymbol{G}_r\boldsymbol{H}_r^T$ (see notations in Eqs. (2.65) and (2.67), too):

$$\begin{aligned} \mathcal{G}(\hat{\boldsymbol{G}}_{\hat{r}}) &= \mathcal{F}(\boldsymbol{F} - \mathcal{A}(\boldsymbol{G}_r\boldsymbol{H}_r^T)), \\ \mathcal{H}(\hat{\boldsymbol{H}}_{\hat{r}}) &= \bar{\mathcal{F}}(\boldsymbol{F} - \mathcal{A}(\boldsymbol{G}_r\boldsymbol{H}_r^T)). \end{aligned} \tag{2.68}$$

This procedure may be applied for an adaptive rank control to reach a final rank, which is as low as possible to get a desired accuracy. However, a computed representation $\boldsymbol{G}_{r+\hat{r}}\boldsymbol{H}_{r+\hat{r}}^T$ is suboptimal in the sense that it does not minimise the expectation of the total potential energy in comparison with all other rank $r + \hat{r}$ representations [158, 157]. The author of [157] proposes a global update of $\boldsymbol{H}_{r+\hat{r}}$ in a successive step that means to compute a new stochastic part $\bar{\boldsymbol{H}}_{r+\hat{r}} := \tau_{\boldsymbol{H}}(\boldsymbol{G}_{r+\hat{r}})$ which results in representation $\boldsymbol{G}_{r+\hat{r}}\bar{\boldsymbol{H}}_{r+\hat{r}}^T$. The global update improves the accuracy of the solution. Nevertheless, the solution usually remains to be suboptimal. (Analogously, a global update of $\boldsymbol{G}_{r+\hat{r}}$ can be done by computing $\bar{\boldsymbol{G}}_{r+\hat{r}} := \tau_{\boldsymbol{G}}(\boldsymbol{H}_{r+\hat{r}})$.)

The restarting algorithm of the GSD [157] is described by Algorithm 2.1. It starts here with empty matrices as an initial setting for $\boldsymbol{G}$ and $\boldsymbol{H}$ and successively applies the mentioned rank update approach introduced by the coupled system in Eq. (2.68). The SI-GSD is applied to solve these equations and is located in lines 5–10 of Algorithm 2.1. Alternatively, the A-GSD may be used. The global update proposed in [157] focuses only on $\boldsymbol{H}$ and, as a consequence, the final result is usually not optimal; it is optionally embedded and located in lines 15–17. The orthonormalisation of the geometrical part $\hat{\boldsymbol{G}}_{\hat{r}_i}$ of the rank-$\hat{r}_i$ update in iteration i provides numerical stability for the scheme. An additional orthonormalisation of the corresponding stochastic part $\hat{\boldsymbol{H}}_{\hat{r}_i}$ is not necessary, and does not have any influence on the final low-rank representation. The i_{max} many updates of rank $\hat{r}_i$ result in the low-rank representation $\boldsymbol{G}\boldsymbol{H}^T$ of rank $\sum_{i=1}^{i_{max}} \hat{r}_i$.

Alternatively to [158, 157, 156] a low-rank approach with successive rank-one updates, which is directly derived from the minimisation problem in Eq. (2.66), was introduced in our technical report [111] under the name VLR-SR1U and is discussed in Section 2.7.2. The VLR-SR1U algorithm is equivalent to the restarting algorithm of

Algorithm 2.1 Restarting algorithm of the GSD applying SI-GSD and optionally a global update

$\boldsymbol{H} \leftarrow \emptyset, \quad \boldsymbol{G} \leftarrow \emptyset, \quad \hat{\boldsymbol{F}} \leftarrow \boldsymbol{F}$
for $i = 1$ to i_{max} **do**
  $\hat{\boldsymbol{F}} \leftarrow \hat{\boldsymbol{F}} - \mathcal{A}(\boldsymbol{G}\boldsymbol{H}^T)$

  $\hat{\boldsymbol{H}}_{\hat{r}_i} \leftarrow$ init, $\quad \hat{\boldsymbol{G}}_{\hat{r}_i} \leftarrow \emptyset$
  while not accurate enough and max. number of iterations not reached **do**
    $\hat{\boldsymbol{G}}_{\hat{r}_i} \leftarrow$ solve $\mathcal{G}(\hat{\boldsymbol{G}}_{\hat{r}_i}) = \mathcal{F}(\hat{\boldsymbol{F}})$
    $\hat{\boldsymbol{G}}_{\hat{r}_i} \leftarrow$ orthonormalise $\hat{\boldsymbol{G}}_{\hat{r}_i}$
    $\hat{\boldsymbol{H}}_{\hat{r}_i} \leftarrow$ solve $\mathcal{H}(\hat{\boldsymbol{H}}_{\hat{r}_i}) = \bar{\mathcal{F}}(\hat{\boldsymbol{F}})$
  end while

  $\boldsymbol{G} \leftarrow [\boldsymbol{G}, \hat{\boldsymbol{G}}_{\hat{r}_i}]$
  $\boldsymbol{H} \leftarrow [\boldsymbol{H}, \hat{\boldsymbol{H}}_{\hat{r}_i}]$

  if global update **then**
    $\boldsymbol{H} \leftarrow \tau_{\boldsymbol{H}}(\boldsymbol{G})$
  end if
end for

the GSD (see Algorithm 2.1) with the setting $\hat{r}_i = 1$ ($\forall i \in \{1, \ldots, i_{max}\}$) and without a global update. A global update is also presented for the VLR-SR1U scheme, see Section 2.7.2.2. It is based on rank-one update matrices and updates both the stochastic and geometrical parts of a current low-rank approximation in an alternating iterative process. In contrast to the restarting algorithm of the GSD, the VLR-SR1U scheme enables one to reach optimal low-rank representations by its global update. The global update scheme is derived by applying the VLR-SR1U scheme onto projections of the original system to be solved. Additionally, the VLR-SR1U scheme optionally provides to construct the solution space adaptively, see Section 2.7.2.3.

2.7.2 Variational Low-Rank Approach with Successive Rank-One Updates

The schemes presented in this section are contributions of this thesis and were introduced in our technical report [111]. The minimisation problem in Eq. (2.66) was a motivation to directly derive a successive rank-one update scheme, which

is named the basic variational low-rank approach with successive rank-one updates (VLR-SR1U scheme). As already mentioned, this scheme provides suboptimal approximations [158] and is a greedy one [200], for which the convergence is already proved in [7] for linear systems of full rank. A rank-one update is performed by alternating iterations, that means in an iteration one part of the rank-one update is solved while the other is fixed to the current state and then vice versa. That is comparable to the alternating least squares algorithm [22, 21]. The basic VLR-SR1U scheme is applied onto projections of the considered system to be solved to obtain an optimisation scheme — the variational low-rank optimisation (VLR-OPT) scheme also introduced in [111] — for current rank-r approximations. It is an independent scheme, which is for instance applied to optimise low-rank approximations currently obtained by the basic VLR-SR1U scheme on the fly or at a final rank. If the operator of the linear system is linear in the material parameter, then a special — more efficient — handling is proposed. Furthermore, the proposed residual-based solution space estimator (RBSSE scheme) — also in [111] — rates how much a stochastic polynomial which was not yet used contributes for a description of the solution. It is based on an extended residual and is applied in the basic VLR-SR1U scheme to adaptively construct the solution space. The VLR-SR1U scheme is (with or without an application of VLR-OPT) very similar to the PGD methods [159, 42, 41], and similarly the resulting separated representation is not necessarily orthogonal.

The basic VLR-SR1U scheme is comparable to a special configuration of the restarting algorithm, see Algorithm 2.1 in Section 2.7.1. However, the VLR-OPT scheme takes into account the back coupling of both the geometrical and the stochastic parts of the low-rank approximation. The basic VLR-SR1U scheme and the VLR-OPT scheme are derived in Sections 2.7.2.1 and 2.7.2.2. The RBSSE scheme and its application is discussed in 2.7.2.3.

Extensive numerical experiments associated with these schemes are presented in Section 4.1.4. An additional application of the basic VLR-SR1U scheme and the VLR-OPT scheme can be found in Section 4.2.2.4.

2.7.2.1 The Basic VLR-SR1U Scheme

The basic VLR-SR1U scheme from our technical report [111] is derived from the minimisation problem in Eq. (2.66) for the stochastic stationary groundwater flow problem discretised by the SGM. At this, the ansatz

$$\boldsymbol{U} = \boldsymbol{U}^{-} + \boldsymbol{g}\boldsymbol{h}^{T} \tag{2.69}$$

describes the rank-one update of a given approximation $\boldsymbol{U}^-$ by the dyadic product of the geometrical update vector $\boldsymbol{g}$ and the stochastic update vector $\boldsymbol{h}$. Differentiating the operator $\mathcal{E}$ in Eq. (2.66) with respect to $\boldsymbol{U}$ means

$$\frac{\partial \mathcal{E}(\boldsymbol{U})}{\partial \boldsymbol{U}}(\delta \boldsymbol{U}) \;=\; \underbrace{(\mathcal{A}(\boldsymbol{U}) \;-\; \boldsymbol{F})}_{=:R(\boldsymbol{U})} \;:\; \delta \boldsymbol{U}. \tag{2.70}$$

This allows to consider the perturbation of $\boldsymbol{U}$ by varying once $\boldsymbol{g}$ and once $\boldsymbol{h}$:

$$\begin{aligned} \frac{\partial \boldsymbol{U}}{\partial \boldsymbol{g}}(\delta \boldsymbol{g}) &= \delta \boldsymbol{g} \boldsymbol{h}^T \\ \frac{\partial \boldsymbol{U}}{\partial \boldsymbol{h}}(\delta \boldsymbol{h}) &= \boldsymbol{g} \delta \boldsymbol{h}^T. \end{aligned}$$

Then, the derivatives of $\mathcal{E}$ with respect to $\boldsymbol{g}$ and $\boldsymbol{h}$ are given by

$$\begin{aligned} \frac{\partial \mathcal{E}(\boldsymbol{U})}{\partial \boldsymbol{g}}(\delta \boldsymbol{g}) \;&=\; R_{\boldsymbol{U}} : (\delta \boldsymbol{g} \boldsymbol{h}^T) \;=\; \delta \boldsymbol{g}^T R_{\boldsymbol{U}} \boldsymbol{h} \\ \frac{\partial \mathcal{E}(\boldsymbol{U})}{\partial \boldsymbol{h}}(\delta \boldsymbol{h}) \;&=\; R_{\boldsymbol{U}} : (\boldsymbol{g} \delta \boldsymbol{h}^T) \;=\; \boldsymbol{g}^T R_{\boldsymbol{U}} \delta \boldsymbol{h} \end{aligned} \tag{2.71}$$

with $R_{\boldsymbol{U}} := R(\boldsymbol{U})$. The mentioned minimisation problem is expressed by $\frac{\partial \mathcal{E}(\boldsymbol{U})}{\partial \boldsymbol{g}}(\delta \boldsymbol{g}) \equiv 0$ and $\frac{\partial \mathcal{E}(\boldsymbol{U})}{\partial \boldsymbol{h}}(\delta \boldsymbol{h}) \equiv 0$. As $\delta \boldsymbol{g}$ and $\delta \boldsymbol{h}$ are arbitrary it follows

$$\begin{aligned} R_{\boldsymbol{U}} \boldsymbol{h} \;&=\; \boldsymbol{0} \\ \boldsymbol{g}^T R_{\boldsymbol{U}} \;&=\; \boldsymbol{0}. \end{aligned} \tag{2.72}$$

Substituting $\boldsymbol{U}$ by the ansatz $\boldsymbol{U}_{r+1} = \boldsymbol{U}_r + \boldsymbol{g}_{r+1} \boldsymbol{h}_{r+1}^T$ with rank r, inserting this ansatz into Eq. (2.72), and arranging the resulting equations leads to the coupled system to be solved:

$$\begin{aligned} &\boldsymbol{P}\, \boldsymbol{g}_{r+1} \;=\; \boldsymbol{b} \\ &\quad \text{with} \quad \boldsymbol{P} := \sum_{i=0}^{l} \underbrace{\boldsymbol{h}_{r+1}^T \boldsymbol{\Delta}_i \boldsymbol{h}_{r+1}}_{=:d_i} \boldsymbol{K}_i, \quad \boldsymbol{b} := (\boldsymbol{F} - \mathcal{A}(\boldsymbol{U}_r))\, \boldsymbol{h}_{r+1}, \\ &\boldsymbol{Q}\, \boldsymbol{h}_{r+1} \;=\; \bar{\boldsymbol{b}} \\ &\quad \text{with} \quad \boldsymbol{Q} := \sum_{i=0}^{l} \underbrace{\boldsymbol{g}_{r+1}^T \boldsymbol{K}_i \boldsymbol{g}_{r+1}}_{=:k_i} \boldsymbol{\Delta}_i, \quad \bar{\boldsymbol{b}} := (\boldsymbol{F} - \mathcal{A}(\boldsymbol{U}_r))^T\, \boldsymbol{g}_{r+1}, \end{aligned} \tag{2.73}$$

where $\boldsymbol{U}_r$ is of course represented in its sum of dyadic products: $\boldsymbol{U}_r = \sum_{i=1}^{r} \boldsymbol{g}_i\, \boldsymbol{h}_i^T = \boldsymbol{G}\boldsymbol{H}^T$.

If the stiffness matrix $\boldsymbol{K}_i = \boldsymbol{K}(\kappa_i)$ is linear with respect to its material κ_i (as in Eq. (2.34)) the coefficient matrix $\boldsymbol{P}$ can be rewritten, which significantly reduces the computational costs:

$$\boldsymbol{P} = \sum_{i=0}^{l} d_i \boldsymbol{K}(\kappa_i) \;=\; \boldsymbol{K}(\Sigma_{i=0}^{l} d_i \kappa_i). \tag{2.74}$$

This more efficient handling is not the only advantage. The sum of the material parameter realisations satisfy a positive definite structure. In contrast, a single material parameter κ_i ($i \in \{0, \ldots, l\}$) may be negative. When a simulation code — available as a black box — is involved to construct a stiffness matrix for a negative material realisation the code may crash or stop working.

The coupled system in Eq. (2.73) may be solved by an alternating iterative process, described in Algorithm 2.2. Empty matrices are chosen as an initial setting for $\boldsymbol{G}$ and $\boldsymbol{H}$, but for example the deterministic solution corresponding to the expectation of the uncertain parameter may be used for an alternative initial setting. The first loop — specified in lines 2–12 of Algorithm 2.2 — increments the rank. The initial guess for the geometrical part of a current rank-one update cannot be chosen as a zero vector, as otherwise the iterative process would stagnate. Here, the guess is chosen to be random. The second loop — specified in lines 4–9 — performs the alternating iterations to compute the current rank-one update. The iterating vectors are simply normalised to keep them bounded. The alternating iterative process is comparable to the alternating least squares algorithm [22, 21].

Algorithm 2.2 Basic VLR-SR1U

1: $\boldsymbol{H} \leftarrow \emptyset, \quad \boldsymbol{G} \leftarrow \emptyset$
2: **while** not accurate enough and max. number of iterations not reached **do**
3: $\quad \boldsymbol{h} \leftarrow$ rand, $\quad \boldsymbol{g} \leftarrow \boldsymbol{0}$
4: $\quad$ **while** not accurate enough and max. number of iterations not reached **do**
5: $\quad\quad \boldsymbol{h} \leftarrow$ normalise $\boldsymbol{h}$
6: $\quad\quad \boldsymbol{g} \leftarrow$ solve $\boldsymbol{P}\,\boldsymbol{g} = \boldsymbol{b}$
7: $\quad\quad \boldsymbol{g} \leftarrow$ normalise $\boldsymbol{g}$
8: $\quad\quad \boldsymbol{h} \leftarrow$ solve $\boldsymbol{Q}\,\boldsymbol{h} = \bar{\boldsymbol{b}}$
9: $\quad$ **end while**
10: $\quad \boldsymbol{G} \leftarrow [\boldsymbol{G}, \boldsymbol{g}]$
11: $\quad \boldsymbol{H} \leftarrow [\boldsymbol{H}, \boldsymbol{h}]$
12: **end while**

The break criterion of the loop specified in lines 4–9 of Algorithm 2.2 is met when

a maximum number of iterations is reached or the error of the current approximation of the rank-one update falls below a threshold. Before the corresponding error estimation is formulated another notation is introduced for the coefficient matrices and the right-hand sides of Eq. (2.73): the dependency of $\boldsymbol{P}$ on $\boldsymbol{h}_{r+1}$ is expressed by the new notation $\boldsymbol{P}_{\boldsymbol{h}_{r+1}}$ for $\boldsymbol{P}$. Analogously, $\boldsymbol{b}_{\boldsymbol{h}_{r+1}}$, $\boldsymbol{Q}_{\boldsymbol{g}_{r+1}}$, and $\bar{\boldsymbol{b}}_{\boldsymbol{g}_{r+1}}$ replace $\boldsymbol{b}$, $\boldsymbol{Q}$, and $\bar{\boldsymbol{b}}$. The index of the current iteration in the loop is considered additionally: $\boldsymbol{g}_{r+1}$ and $\boldsymbol{h}_{r+1}$ in iteration j are expressed by $\boldsymbol{g}_{r+1}^{(j)}$ and $\boldsymbol{h}_{r+1}^{(j)}$. Then the estimated error of the rank-one update at the end of iteration j is defined by $\epsilon_{\boldsymbol{g}\boldsymbol{h}^T} := \sqrt{\epsilon_{\boldsymbol{g}}^2 + \epsilon_{\boldsymbol{h}}^2}$ with

$$
\begin{aligned}
\epsilon_{\boldsymbol{g}} &:= \| \boldsymbol{P}_{\boldsymbol{h}_{r+1}^{(j-1)}} \boldsymbol{g}_{r+1}^{(j-1)} - \boldsymbol{b}_{\boldsymbol{h}_{r+1}^{(j-1)}} \|_2, \\
\epsilon_{\boldsymbol{h}} &:= \| \boldsymbol{Q}_{\boldsymbol{g}_{r+1}^{(j)}} \boldsymbol{h}_{r+1}^{(j-1)} - \bar{\boldsymbol{b}}_{\boldsymbol{g}_{r+1}^{(j)}} \|_2.
\end{aligned}
\tag{2.75}
$$

Thus, outputs of iterations $j-1$ and j are used to estimate the errors $\epsilon_{\boldsymbol{g}}$ and $\epsilon_{\boldsymbol{h}}$ of $\boldsymbol{g}_{r+1}^{(j)}$ and $\boldsymbol{h}_{r+1}^{(j)}$.

The estimated error $\epsilon_{\boldsymbol{G}\boldsymbol{H}^T}$ of a current low-rank approximation $\boldsymbol{G}\boldsymbol{H}^T$ for the break criterion in line 2 can be obtained by the Euclidean norm (or, more sophisticated, a problem-related norm) of the residual: $\epsilon_{\boldsymbol{G}\boldsymbol{H}^T} := \|R(\boldsymbol{G}\boldsymbol{H}^T)\|_2$ (see Eq. (2.70)).

A rank-one update $\boldsymbol{g}\boldsymbol{h}^T$ for a solution $\boldsymbol{U}$ is orthogonal to the residual $R(\boldsymbol{U})$ of the original system to be solved (see Eq. (2.72)), but it is not necessarily orthogonal to $\boldsymbol{U}$ with respect to the energy norm, that means it usually holds that $\mathcal{A}(\boldsymbol{U}) : \boldsymbol{g}\boldsymbol{h}^T \approx 0$. In other words, a low-rank representation obtained by the basic VLR-SR1U scheme is suboptimal in the sense that it is not the minimiser of the expectation of the total potential energy in comparison with all other representations of the same rank (like it is the case for the restarting algorithm of the GSD [158, 157]). That is obvious in the numerical experiments of Section 4.1.4.1. According to the opinion of the author of this thesis the suboptimality is traced back to the fact that the first equation of the coupled system in Eq. (2.73) to be solved is nonlinear in $\boldsymbol{h}$ and the second equation is nonlinear in $\boldsymbol{g}$. The problem of the suboptimality is addressed in the next section by an optimisation process.

2.7.2.2 The VLR-OPT Scheme for an Optimisation

The VLR-OPT scheme from our technical report [111] optimises a given approximation of rank r for the solution of Eq. (2.65) through alternating iterations on the geometrical and stochastic subspaces, so that the approximation becomes the minimiser of the expectation of the total potential energy in comparison with all other

representations of the same rank. It applies the algorithm of the basic VLR-SR1U scheme on projections of the system (see Eq. (2.65)) to be solved to obtain rank-one matrices to update the geometrical part $\boldsymbol{G}$ and the stochastic part $\boldsymbol{H}$ of an approximation $\boldsymbol{U} = \boldsymbol{G}\boldsymbol{H}^T$ of rank r (see Eq. (2.31)). These rank-one matrices are simply added to the parts (that means, an update does not increase the rank). The VLR-OPT scheme is applied to optimise current low-rank approximations in the basic VLR-SR1U scheme on the fly or at a final rank.

The idea of the VLR-OPT scheme is to determine rank-one matrices $\boldsymbol{g}\boldsymbol{v}^T$ and $\boldsymbol{h}\boldsymbol{w}^T$ to obtain updates for $\boldsymbol{G}$ and $\boldsymbol{H}$:

$$\boldsymbol{G} = \boldsymbol{G}^- + \boldsymbol{g}\boldsymbol{v}^T \text{ and } \boldsymbol{H} = \boldsymbol{H}^- + \boldsymbol{h}\boldsymbol{w}^T. \tag{2.76}$$

For this purpose the algorithm of the basic VLR-SR1U scheme is applied onto projections of the system (see Eq. (2.65)) to be solved.

When the $\boldsymbol{H}$ of $\boldsymbol{U} = \boldsymbol{G}\boldsymbol{H}^T$ is fixed Eq. (2.66) becomes the following minimisation problem in $\boldsymbol{G}$: $\frac{1}{2}\mathcal{A}(\boldsymbol{U})\boldsymbol{H} : \boldsymbol{G} - \boldsymbol{F}\boldsymbol{H} : \boldsymbol{G} \to \min$. Instead of solving the system in Eq. (2.65) the minimisation problem solves the projection of that system on the column space of $\boldsymbol{H}$:

$$\mathcal{A}(\boldsymbol{U})\boldsymbol{H} \;=\; \boldsymbol{F}\boldsymbol{H} \iff \mathcal{A}_{\boldsymbol{H}}(\boldsymbol{G}) \;=\; \boldsymbol{F}_{\boldsymbol{H}}. \tag{2.77}$$

Analogously to Section 2.7.2.1, the basic VLR-SR1U algorithm can be applied on that projected system to obtain a rank-one update $\boldsymbol{g}\boldsymbol{v}^T$ for $\boldsymbol{G}$. The application leads to the coupled system to be solved:

$$\begin{aligned}
&\boldsymbol{P}_{\boldsymbol{H}}\, \boldsymbol{g}_{r+1} = \boldsymbol{b}_{\boldsymbol{H}} \\
&\quad \text{with} \quad \boldsymbol{P}_{\boldsymbol{H}} := \sum_{i=0}^{l} \underbrace{\boldsymbol{v}_{r+1}^T \boldsymbol{H}^T \boldsymbol{\Delta}_i \boldsymbol{H} \boldsymbol{v}_{r+1}}_{=:\bar{d}_i \in \mathbb{R}} \boldsymbol{K}_i \\
&\quad \text{and} \quad \boldsymbol{b}_{\boldsymbol{H}} := \left(\boldsymbol{F}_{\boldsymbol{H}} - \mathcal{A}_{\boldsymbol{H}}(\boldsymbol{G})\right) \boldsymbol{v}_{r+1}, \\
&\boldsymbol{Q}_{\boldsymbol{H}}\, \boldsymbol{v}_{r+1} = \bar{\boldsymbol{b}}_{\boldsymbol{H}} \\
&\quad \text{with} \quad \boldsymbol{Q}_{\boldsymbol{H}} := \sum_{i=0}^{l} \underbrace{\boldsymbol{g}_{r+1}^T \boldsymbol{K}_i \boldsymbol{g}_{r+1}}_{=:k_i \in \mathbb{R}} \boldsymbol{H}^T \boldsymbol{\Delta}_i \boldsymbol{H} \\
&\quad \text{and} \quad \bar{\boldsymbol{b}}_{\boldsymbol{H}} := \left(\boldsymbol{F}_{\boldsymbol{H}} - \mathcal{A}_{\boldsymbol{H}}(\boldsymbol{G})\right)^T \boldsymbol{g}_{r+1}.
\end{aligned} \tag{2.78}$$

In the same manner, the basic VLR-SR1U algorithm can be applied onto the projection of the system in Eq. (2.65) on the column space of $\boldsymbol{G}$:

$$\boldsymbol{G}^T \mathcal{A}(\boldsymbol{U}) \;=\; \boldsymbol{G}^T \boldsymbol{F} \iff \mathcal{A}_{\boldsymbol{G}}(\boldsymbol{H}) \;=\; \boldsymbol{F}_{\boldsymbol{G}}. \tag{2.79}$$

The resulting coupled system which has to be solved to obtain a rank-one update $\boldsymbol{h}\boldsymbol{w}^T$ for $\boldsymbol{H}$ is given by

$$\begin{aligned}
&\boldsymbol{P}_{\boldsymbol{G}}\, \boldsymbol{h}_{r+1} = \boldsymbol{b}_{\boldsymbol{G}} \\
&\quad \text{with} \quad \boldsymbol{P}_{\boldsymbol{G}} := \sum_{i=0}^{l} \underbrace{\boldsymbol{w}_{r+1}^T \boldsymbol{G}^T \boldsymbol{K}_i \boldsymbol{G} \boldsymbol{w}_{r+1}}_{=:\bar{k}_i \in \mathbb{R}} \boldsymbol{\Delta}_i \\
&\quad \text{and} \quad \boldsymbol{b}_{\boldsymbol{G}} := \left(\boldsymbol{F}_{\boldsymbol{G}}^T - \mathcal{A}_{\boldsymbol{G}}^T(\boldsymbol{H})\right) \boldsymbol{w}_{r+1}, \\
&\boldsymbol{Q}_{\boldsymbol{G}}\, \boldsymbol{w}_{r+1} = \bar{\boldsymbol{b}}_{\boldsymbol{G}} \\
&\quad \text{with} \quad \boldsymbol{Q}_{\boldsymbol{G}} := \sum_{i=0}^{l} \underbrace{\boldsymbol{h}_{r+1}^T \boldsymbol{\Delta}_i \boldsymbol{h}_{r+1}}_{=:d_i \in \mathbb{R}} \boldsymbol{G}^T \boldsymbol{K}_i \boldsymbol{G} \\
&\quad \text{and} \quad \bar{\boldsymbol{b}}_{\boldsymbol{G}} := \left(\boldsymbol{F}_{\boldsymbol{G}}^T - \mathcal{A}_{\boldsymbol{G}}^T(\boldsymbol{H})\right)^T \boldsymbol{h}_{r+1}.
\end{aligned} \tag{2.80}$$

Solving the coupled systems in Eqs. (2.78) and (2.80) provides rank-one update matrices for $\boldsymbol{G}$ and $\boldsymbol{H}$, which are then added to the geometrical and stochastic parts $\boldsymbol{G}$ and $\boldsymbol{H}$ of the current approximation of rank r. Consequently, all rank-one update vectors of $\boldsymbol{G}$ and $\boldsymbol{H}$ are updated. The VLR-OPT scheme performs a sequence of such updates to optimise a given approximation of rank r.

The VLR-OPT scheme to optimise a given approximation $\tilde{\boldsymbol{U}} = \tilde{\boldsymbol{G}}\tilde{\boldsymbol{H}}^T$ of rank r is described in Algorithm 2.3. The energy minimisations in lines 3 and 5 exactly mean to solve the coupled systems in Eqs. (2.78) and (2.80). This is done by applying the alternating iterative process of the basic VLR-SR1U algorithm (see Algorithm 2.2) on each of the mentioned coupled systems. That results in the two rank-one update solvers described by Algorithms 2.4 and 2.5. The tensor products and additions in lines 4 and 6 of Algorithm 2.3 are performed to update all rank-one update vectors globally without increasing rank r.

The definition of the break criteria for the two rank-one update solvers in Algorithms 2.4 and 2.5 are analogous to the definition of the break criterion for a rank-one update in the basic VLR-SR1U scheme, see line 4 in Algorithm 2.2 and the corresponding explanations in Section 2.7.2.1. Consequently, a new notation is introduced for the coefficient matrices and right-hand sides of the coupled systems

Algorithm 2.3 VLR-OPT

$\boldsymbol{G} \leftarrow \tilde{\boldsymbol{G}}, \quad \boldsymbol{H} \leftarrow \tilde{\boldsymbol{H}}$
while not accurate enough and max. number of iterations not reached **do**
  $\boldsymbol{g}, \boldsymbol{v} \leftarrow$ minimise $\mathcal{E}((\boldsymbol{G} + \boldsymbol{g}\boldsymbol{v}^T)\boldsymbol{H}^T)$
  $\boldsymbol{G} \leftarrow \boldsymbol{G} + \boldsymbol{g}\boldsymbol{v}^T$
  $\boldsymbol{h}, \boldsymbol{w} \leftarrow$ minimise $\mathcal{E}(\boldsymbol{G}(\boldsymbol{H} + \boldsymbol{h}\boldsymbol{w}^T)^T)$
  $\boldsymbol{H} \leftarrow \boldsymbol{H} + \boldsymbol{h}\boldsymbol{w}^T$
end while

Algorithm 2.4 minimise $\mathcal{E}((\boldsymbol{G} + \boldsymbol{g}\boldsymbol{v}^T)\boldsymbol{H}^T)$

$\boldsymbol{g} \leftarrow$ rand, $\quad \boldsymbol{v} \leftarrow \boldsymbol{0}$
while not accurate enough and max. number of iterations not reached **do**
  $\boldsymbol{g} \leftarrow$ normalise $\boldsymbol{g}$
  $\boldsymbol{v} \leftarrow$ solve $\boldsymbol{Q}_{\boldsymbol{H}}\, \boldsymbol{v} = \bar{\boldsymbol{b}}_{\boldsymbol{H}}$
  $\boldsymbol{v} \leftarrow$ normalise $\boldsymbol{v}$
  $\boldsymbol{g} \leftarrow$ solve $\boldsymbol{P}_{\boldsymbol{H}}\, \boldsymbol{g} = \boldsymbol{b}_{\boldsymbol{H}}$
end while

Algorithm 2.5 minimise $\mathcal{E}(\boldsymbol{G}(\boldsymbol{H} + \boldsymbol{h}\boldsymbol{w}^T)^T)$

$\boldsymbol{h} \leftarrow$ rand, $\quad \boldsymbol{w} \leftarrow \boldsymbol{0}$
while not accurate enough and max. number of iterations not reached **do**
  $\boldsymbol{h} \leftarrow$ normalise $\boldsymbol{h}$
  $\boldsymbol{w} \leftarrow$ solve $\boldsymbol{Q}_{\boldsymbol{G}}\, \boldsymbol{w} = \bar{\boldsymbol{b}}_{\boldsymbol{G}}$
  $\boldsymbol{w} \leftarrow$ normalise $\boldsymbol{w}$
  $\boldsymbol{h} \leftarrow$ solve $\boldsymbol{P}_{\boldsymbol{G}}\, \boldsymbol{h} = \boldsymbol{b}_{\boldsymbol{G}}$
end while

in Eqs. (2.78) and (2.80): $\boldsymbol{P}_{\boldsymbol{H},\boldsymbol{v}_{r+1}^{(j)}}$ expresses that $\boldsymbol{P}_{\boldsymbol{H}}$ of Eq. (2.78) is defined by $\boldsymbol{v}_{r+1}$ obtained in iteration j of the loop in Algorithm 2.4. Analogously, further notations are $\boldsymbol{b}_{\boldsymbol{H},\boldsymbol{v}_{r+1}^{(j)}}$, $\boldsymbol{Q}_{\boldsymbol{H},\boldsymbol{g}_{r+1}^{(j)}}$, and $\bar{\boldsymbol{b}}_{\boldsymbol{H},\boldsymbol{g}_{r+1}^{(j)}}$ for the coupled system in Eq. (2.78) and $\boldsymbol{P}_{\boldsymbol{G},\boldsymbol{w}_{r+1}^{(j)}}$, $\boldsymbol{b}_{\boldsymbol{G},\boldsymbol{w}_{r+1}^{(j)}}$, $\boldsymbol{Q}_{\boldsymbol{G},\boldsymbol{h}_{r+1}^{(j)}}$, and $\bar{\boldsymbol{b}}_{\boldsymbol{G},\boldsymbol{h}_{r+1}^{(j)}}$ for the coupled system in Eq. (2.80). The estimated error of a rank-one update at the end of iteration j in Algorithm 2.4 is defined by $\epsilon_{\boldsymbol{g}\boldsymbol{v}^T} := \sqrt{\epsilon_{\boldsymbol{g}}^2 + \epsilon_{\boldsymbol{v}}^2}$ with

$$
\begin{aligned}
\epsilon_{\boldsymbol{g}} &:= \| \boldsymbol{P}_{\boldsymbol{H},\boldsymbol{v}_{r+1}^{(j-1)}} \boldsymbol{g}_{r+1}^{(j-1)} - \boldsymbol{b}_{\boldsymbol{H},\boldsymbol{v}_{r+1}^{(j-1)}} \|_2, \\
\epsilon_{\boldsymbol{v}} &:= \| \boldsymbol{Q}_{\boldsymbol{H},\boldsymbol{g}_{r+1}^{(j)}} \boldsymbol{v}_{r+1}^{(j-1)} - \bar{\boldsymbol{b}}_{\boldsymbol{H},\boldsymbol{g}_{r+1}^{(j)}} \|_2.
\end{aligned}
\tag{2.81}
$$

Accordingly, the estimated error of a rank-one update at the end of iteration j in Algorithm 2.5 is defined by $\epsilon_{\boldsymbol{h}\boldsymbol{w}^T} := \sqrt{\epsilon_{\boldsymbol{h}}^2 + \epsilon_{\boldsymbol{w}}^2}$ with

$$\begin{aligned} \epsilon_{\boldsymbol{h}} &:= \| \boldsymbol{P}_{\boldsymbol{G},\boldsymbol{w}_{r+1}^{(j-1)}} \boldsymbol{h}_{r+1}^{(j-1)} - \boldsymbol{b}_{\boldsymbol{G},\boldsymbol{w}_{r+1}^{(j-1)}} \|_2, \\ \epsilon_{\boldsymbol{w}} &:= \| \boldsymbol{Q}_{\boldsymbol{G},\boldsymbol{h}_{r+1}^{(j)}} \boldsymbol{w}_{r+1}^{(j-1)} - \bar{\boldsymbol{b}}_{\boldsymbol{G},\boldsymbol{h}_{r+1}^{(j)}} \|_2. \end{aligned} \tag{2.82}$$

The estimated error $\epsilon_{\boldsymbol{G}\boldsymbol{H}^T}$ of $\boldsymbol{G}\boldsymbol{H}^T$ at the end of an iteration in the loop of Algorithm 2.3 is defined by $\epsilon_{\boldsymbol{G}\boldsymbol{H}^T} := \sqrt{\epsilon_{\boldsymbol{G}}^2 + \epsilon_{\boldsymbol{H}}^2}$, where the estimated errors $\epsilon_{\boldsymbol{G}}$ and $\epsilon_{\boldsymbol{H}}$ are obtained by the Euclidean norm of the residuals of Eqs. (2.77) and (2.79):

$$\begin{aligned} \epsilon_{\boldsymbol{G}} &:= \| \mathcal{A}_{\boldsymbol{H}}(\boldsymbol{G}) - \boldsymbol{F}_{\boldsymbol{H}} \|_2, \\ \epsilon_{\boldsymbol{H}} &:= \| \mathcal{A}_{\boldsymbol{G}}(\boldsymbol{H}) - \boldsymbol{F}_{\boldsymbol{G}} \|_2. \end{aligned} \tag{2.83}$$

The basic VLR-SR1U scheme is extended by an application of the VLR-OPT scheme. This is done in different ways. In one configuration VLR-SR1U uses VLR-OPT after each rank-one update to optimise the current low-rank approximation on the fly. VLR-OPT is then located between lines 11 and 12 of Algorithm 2.2. This configuration is referred to as *VLR-SR1U-OPT(1)*. Another configuration uses the VLR-OPT scheme only as a post-processor, that means at the final rank. In that case VLR-OPT is located after the last line 12 of Algorithm 2.2. That is referred to as *VLR-SR1U-OPT(2)*. Table 2.2 summarises the two configurations.

name	configuration
VLR-SR1U-OPT(1)	VLR-SR1U + VLR-OPT after each rank-one update
VLR-SR1U-OPT(2)	VLR-SR1U + VLR-OPT only at the final rank

Table 2.2: *Used combinations of the basic VLR-SR1U scheme and the VLR-OPT scheme.*

2.7.2.3 The RBSSE Scheme for an Adaptive Solution Space Construction

A low-rank approach provides a possibility to reduce the computational costs and memory usage. Advantages of the same kind can be additionally obtained if a small

solution space can be found, which represents the solution well. A residual-based solution space estimator (RBSSE) is proposed, which tries to identify relevant stochastic polynomials for the description of the solution and was introduced in our technical report [111]. It considers an extended residual corresponding to the current approximation of the solution and stochastic polynomials not yet used. This is similar to [143] and in the main idea comparable to [56]. The residual is transferred to an indicator which has the same unit of measurement as the solution, so that the relevant stochastic polynomials can be identified. The RBSSE scheme is applied in the basic VLR-SR1U scheme to adaptively construct the solution space [111].

The RBSSE scheme uses the residual of the discretised problem with respect to the current approximation of the solution. It is explained in the following by means of the SGM-discretised groundwater flow problem in Eq. (2.65). The corresponding residual is defined as

$$\boldsymbol{R} \;:=\; \boldsymbol{F} \;-\; \sum_{i=0}^{l} \boldsymbol{K}_i \, \boldsymbol{U} \, \boldsymbol{\Delta}_i. \tag{2.84}$$

The set of the current stochastic polynomials $\{\psi_\alpha\}_{\alpha\in\mathcal{I}^c}$ ($|\{\psi_\alpha\}_{\alpha\in\mathcal{I}^c}| = N_S$) is now extended by additional ones $\{\psi_{\alpha^+}\}_{\alpha^+\in\mathcal{I}^{c+}}$ ($|\{\psi_{\alpha^+}\}_{\alpha^+\in\mathcal{I}^{c+}}| = N_S^+$). The influence of these new stochastic polynomials on the quality of the solution is estimated by artificially extending the residual. For this purpose each stochastic matrix $\boldsymbol{\Delta}_i$ results in:

$$\boldsymbol{\Delta}_i^{\oplus} := \left(\begin{array}{c|c} \boldsymbol{\Delta}_i & \boldsymbol{\Delta}_i^{\triangleleft} \\ \hline \boldsymbol{\Delta}_i^{\triangleright} & \boldsymbol{\Delta}_i^{+} \end{array} \right).$$

The matrix $\boldsymbol{\Delta}_i^{+}$ only captures the additional polynomials. The matrices $\boldsymbol{\Delta}_i^{\triangleleft}$ and $\boldsymbol{\Delta}_i^{\triangleright} = (\boldsymbol{\Delta}_i^{\triangleleft})^T$ describe the coupling between the current and the additional stochastic polynomials. The new matrices are defined by $[\boldsymbol{\Delta}_i^{+}]_{\alpha^+,\beta^+} := \sum_{\gamma\in\mathcal{I}^c} \sqrt{\lambda_i}\, \xi_{i,\gamma}\, [\boldsymbol{\Delta}_\gamma]_{\alpha^+,\beta^+}$ and $[\boldsymbol{\Delta}_i^{\triangleleft}]_{\alpha^+,\beta} := \sum_{\gamma\in\mathcal{I}^c} \sqrt{\lambda_i}\, \xi_{i,\gamma}\, [\boldsymbol{\Delta}_\gamma]_{\alpha^+,\beta}$ (see Eq. (2.64)). Matrix $\boldsymbol{\Delta}_i^{\oplus}$ allows to consider the extended residual:

$$\boldsymbol{R}^{\oplus} \;:=\; \boldsymbol{F}^{\oplus} \;-\; \sum_{i=0}^{l} \boldsymbol{K}_i \, \boldsymbol{U}^{\oplus} \, \boldsymbol{\Delta}_i^{\oplus}.$$

The coefficients of the extended solution $\boldsymbol{U}^{\oplus}$ associated with the added stochastic polynomials are set to zero: $\boldsymbol{U}^{\oplus} := [\boldsymbol{U}, \boldsymbol{0}]$ with a N_X-by-N_S^+ zero matrix $\boldsymbol{0}$. The extended right-hand side (RHS) $\boldsymbol{F}^{\oplus}$ results from the concatenation of $\boldsymbol{F}$ and $\boldsymbol{F}^{+}$: $\boldsymbol{F}^{\oplus} := [\boldsymbol{F}, \boldsymbol{F}^{+}]$. $\boldsymbol{F}^{+}$ is the RHS corresponding to the added stochastic polynomials.

The added zero coefficients of the extended solution allow to simplify the computation of $\boldsymbol{R}^{\oplus}$:

$$\boldsymbol{R}^{\oplus} \;:=\; \boldsymbol{F}^{\oplus} \;-\; \sum_{i=0}^{l} \boldsymbol{K}_i \, \boldsymbol{U} \, \boldsymbol{\Delta}_i^{\uplus} \qquad \text{with} \qquad \boldsymbol{\Delta}_i^{\uplus} := [\boldsymbol{\Delta}_i, \boldsymbol{\Delta}_i^{\triangleleft}].$$

$\boldsymbol{R}^{\oplus}$ may be considered as the concatenation of $\boldsymbol{R}$ and $\boldsymbol{R}^{+}$: $\boldsymbol{R}^{\oplus} = [\boldsymbol{R}, \boldsymbol{R}^{+}]$. $\boldsymbol{R}^{+}$ is the residual associated with the additional stochastic polynomials: $\boldsymbol{R}^{+} := \boldsymbol{F}^{+} - \sum_{i=0}^{l} \boldsymbol{K}_i \, \boldsymbol{U} \, \boldsymbol{\Delta}_i^{\triangleleft}$, $\boldsymbol{R}^{+} \in \mathbb{R}^{N_X \times N_S^{+}}$.

$\boldsymbol{R}^{+}$ is used to interpret the rating of the additional stochastic polynomials. However, $\boldsymbol{R}^{+}$ is not directly used for that purpose, as — physically speaking — $\boldsymbol{R}^{+}$ does not have the same unit of measurement as the solution. Therefore, it may not fulfil the conditions of an "ideal" rating indicator regarding the accuracy of the solution. An indicator $\mathbf{i}$ of the same unit of measurement as the solution can be obtained by the product:

$$\mathbf{i} \;:=\; (\boldsymbol{A}^{+})^{-1} \, \boldsymbol{r}^{+} \;\text{ with }\; \boldsymbol{A}^{+} := \sum_{i=0}^{l} \boldsymbol{\Delta}_i^{+} \otimes \boldsymbol{K}_i, \tag{2.85}$$

see Eq. (2.63). $\boldsymbol{r}^{+} \in \mathbb{R}^{N_X \cdot N_S^{+}}$ is the proper vectorisation of matrix $\boldsymbol{R}^{+}$. In practice the symmetric matrix $(\boldsymbol{A}^{+})^{-1}$ is not computed. The indicator should be combined here with schemes which emphasise performance and, as a consequence, it should be computed with little costs. Instead of using $(\boldsymbol{A}^{+})^{-1}$, the matrix

$$\boldsymbol{J} \;:=\; \underbrace{\left(\operatorname{diag}(\boldsymbol{A}^{+})\right)}_{=:\boldsymbol{D}}^{-1}$$

is used known as the Jacobi preconditioner in the context of preconditioning [74]. The matrix $\boldsymbol{D}$ is a diagonal matrix with the diagonal of $\boldsymbol{A}^{+}$ and consequently easily invertible. Its computation can be simply performed by

$$[\boldsymbol{D}]_{j,k} = \sum_{i=0}^{l} [\boldsymbol{\Delta}_i^{+}]_{k,k} \, [\boldsymbol{K}_i]_{j,j},$$

where $[\boldsymbol{B}]_{j,k}$ marks the entry of a given matrix $\boldsymbol{B}$ at the j'th row and the k'th column. The hereby obtained indicator $\mathbf{i}_{\boldsymbol{J}} := \boldsymbol{J} \, \boldsymbol{r}^{+}$ can also be written in matrix notation

$$\mathfrak{I}_{\boldsymbol{J}} \;:=\; \hat{\boldsymbol{J}} \odot \boldsymbol{R}^{+},$$

where $\hat{\boldsymbol{J}}$ is the proper matrix notation of the diagonal of $\boldsymbol{J}$, and $\odot$ is the Hadamard product [85] for matrices.

For each stochastic polynomial the indicator $\mathfrak{I}_{\boldsymbol{J}}$ contains as many values as geometrical DoFs. These values are reduced to one value by the Euclidean norm to rate the corresponding stochastic polynomials. This is reflected by the function $r(\mathfrak{I}_{\boldsymbol{J}}) := [\ \|(\mathfrak{I}_{\boldsymbol{J}})_1\|_2\ , \ldots,\ \|(\mathfrak{I}_{\boldsymbol{J}})_{N_S^+}\|_2\ ] \in \mathbb{R}^{N_S^+}$, where $(\mathfrak{I}_{\boldsymbol{J}})_j$ means the j'th column of matrix $\mathfrak{I}_{\boldsymbol{J}}$.

Algorithm 2.6 describes the RBSSE scheme. The list of required input arguments is not complete but shows the essential ones. There are different possibilities to define the preconditioner, the reduction function r, or the selection of relevant additional stochastic polynomials in line 7. The Jacobi preconditioner and the previously defined reduction function r are chosen here. The selection function could for instance select 10% of the best rated additional stochastic polynomials.

Algorithm 2.6 RBSSE

Require: $\mathcal{I}^c, \mathcal{I}^{c^+}, \boldsymbol{F}^+$
1: $\{\boldsymbol{\Delta}_i^{\triangleleft}\}_{i\in\{0,\ldots,l\}} \leftarrow$ construct($\mathcal{I}^c, \mathcal{I}^{c^+}$)
2: $\{\boldsymbol{\Delta}_i^{+}\}_{i\in\{0,\ldots,l\}} \leftarrow$ construct($\mathcal{I}^{c^+}$)
3: $\boldsymbol{R}^+ \leftarrow$ compute($\{\boldsymbol{\Delta}_i^{\triangleleft}\}_{i\in\{0,\ldots,l\}}, \boldsymbol{F}^+$)
4: $\hat{\boldsymbol{J}} \leftarrow$ compute($\{\boldsymbol{\Delta}_i^{+}\}_{i\in\{0,\ldots,l\}}$)
5: $\mathfrak{I}_{\hat{\boldsymbol{J}}} \leftarrow$ compute($\boldsymbol{J}, \boldsymbol{R}^+$)
6: $r(\mathfrak{I}_{\boldsymbol{J}}) \leftarrow$ reduce($\mathfrak{I}_{\boldsymbol{J}}$)
7: $\mathcal{I}^+ \leftarrow$ select($r(\mathfrak{I}_{\boldsymbol{J}}), \mathcal{I}^{c^+}$)
8: **return** $\mathcal{I}^+$

The RBSSE scheme is now applied inside Algorithm 2.2 of the basic VLR-SR1U scheme to obtain an adaptive construction of the solution space. That results in Algorithm 2.7 and is referred to as *VLR-SR1U-ADAPT*. The input arguments are given by an initial set $\mathcal{I}^c$ of already accepted stochastic polynomials and a set $\mathcal{I}^{c^+}$ of potentially relevant stochastic polynomials. $\mathcal{I}^{c^+}$ should not be too large as otherwise the rating of the set of additional stochastic polynomials may become too memory- and time consuming. As a consequence and in contrast to Algorithm 2.7, $\mathcal{I}^{c^+}$ may be individually generated for each rank-one update. The set $\mathcal{I}^+$ contains the current best rated stochastic polynomials, which are added to the current set $\mathcal{I}^c$ for the next rank iteration (see line 5). The adaptive construction of the solution space is presented in lines 4–7 of Algorithm 2.7. The system update in line 6 means to extend the entire system to the added stochastic polynomials. The rating and selection of the additional stochastic polynomials happen in lines 19–21 of Algorithm 2.7. Optionally, the VLR-OPT algorithm (see Algorithm 2.3) can be invoked also for instance in line 18 of Algorithm 2.7 for an OPT(1) configuration or after line 22 of

Algorithm 2.7 for an OPT(2) configuration. A combination is also demonstrated in Section 4.1.4.3.

Algorithm 2.7 VLR-SR1U-ADAPT

Require: $\mathcal{I}^c, \mathcal{I}^{c^+}$
$\mathcal{I}^+ \leftarrow \emptyset$
$\boldsymbol{H} \leftarrow \emptyset, \quad \boldsymbol{G} \leftarrow \emptyset$
while not accurate enough and max. number of iterations not reached **do**
  if $\mathcal{I}^+ \neq \emptyset$ **then**
    $\mathcal{I}^c \leftarrow [\mathcal{I}^c, \mathcal{I}^+]$
    update system
  end if

  $\boldsymbol{h} \leftarrow$ rand, $\quad \boldsymbol{g} \leftarrow \boldsymbol{0}$
  while not accurate enough and max. number of iterations not reached **do**
    $\boldsymbol{h} \leftarrow$ normalise $\boldsymbol{h}$
    $\boldsymbol{g} \leftarrow$ solve $\boldsymbol{P}\,\boldsymbol{g} = \boldsymbol{b}$
    $\boldsymbol{g} \leftarrow$ normalise $\boldsymbol{g}$
    $\boldsymbol{h} \leftarrow$ solve $\boldsymbol{Q}\,\boldsymbol{h} = \bar{\boldsymbol{b}}$
  end while
  $\boldsymbol{G} \leftarrow [\boldsymbol{G}, \boldsymbol{g}]$
  $\boldsymbol{H} \leftarrow [\boldsymbol{H}, \boldsymbol{h}]$

  $\boldsymbol{F}^+ \leftarrow$ compute($\mathcal{I}^{c^+}$)
  $\mathcal{I}^+ \leftarrow$ RBSSE($\mathcal{I}^c, \mathcal{I}^{c^+}, \boldsymbol{F}^+$)
  $\mathcal{I}^{c^+} \leftarrow \mathcal{I}^{c^+} \backslash \mathcal{I}^+$
end while

The residual $\boldsymbol{R}^+$ may be primarily used to interpret the rating of the additional polynomials. However, there may be an interplay between the two single residuals of the mentioned composed residual $\boldsymbol{R}^{\oplus} = [\boldsymbol{R}, \boldsymbol{R}^+]$. This is exemplified by considering algorithm VLR-SR1U-ADAPT. The case $\|\boldsymbol{R}\| > \|\boldsymbol{R}^+\|$ (with a suitable matrix norm $\|\cdot\|$) may identify that the basic VLR-SR1U scheme needs to iterate more inside the current rank to reach a more reasonable accuracy before adapting the stochastic solution space to more stochastic polynomials. In contrast $\|\boldsymbol{R}\| < \|\boldsymbol{R}^+\|$ may identify that the chosen additional stochastic polynomials would provide a better accuracy of the solution. In this way a mutual utilisation of the different errors may be conceivable. Certainly, also in this consideration, it may be advantageous to translate the residuals $\boldsymbol{R}$ and $\boldsymbol{R}^+$ to the unit of measurement of the solution — as already done in Eq. (2.85) ff. for $\boldsymbol{R}^+$ — before comparing them.

2.7.2.4 Conclusion

A successive rank-one update scheme — the basic VLR-SR1U scheme [111] — is derived in this section for a stochastic elliptic problem to provide an error controlled low-rank approximation of the solution. It tries to minimise the expectation of the total potential energy of the problem. However, its approximations are suboptimal (which is known for such kind of schemes [158]). A corresponding optimisation for approximations of rank r is provided by the developed VLR-OPT scheme. It is derived by applying the algorithm of the basic VLR-SR1U scheme on projected systems. The VLR-OPT scheme is combined with the basic VLR-SR1U scheme to optimise current approximations on the fly or at a final rank. A further scheme — the RBSSE scheme — is constructed to rate the contribution of not yet used stochastic polynomials for a description of the solution. It is combined with the basic VLR-SR1U scheme to enable an adaptive construction of the solution space.

The numerical behaviour of all these schemes is demonstrated in Section 4.1 on the basis of a stochastic stationary groundwater flow problem. The basic VLR-SR1U scheme and the VLR-OPT scheme are also applied for a laminated composite structure with a linear constitutive law and a partially uncertain material in Section 4.2.

2.8 Conclusion

Different numerical schemes are available for the simulation of probabilistic models. Desired statistics about the solution of the model may be directly formulated by integrals, which are then approximated by numerical integration schemes [37, 124, 126, 123, 71, 170, 52, 99]. Other approaches project the model onto a solution space spanned by stochastic basis functions [145, 122, 11, 153, 66, 216, 20, 26, 208, 62]. The stochastic Galerkin method [73, 128, 12, 217, 127, 3, 57] is such a scheme with promising convergence behaviour, too.

Schemes to find a sparse set of stochastic basis functions [201, 54, 23, 26, 56, 66, 153, 111, 143, 87] or a subspace of low-rank [158, 130, 111, 143, 122, 55, 103] were recently developed and are a vivid area of research. In contrast, combined approaches are still rare [111, 143]. The approach presented here was initially developed in our technical report [111]. It is an SGM-based scheme which provides a successive rank-one update, global optimisation, and an adaptive construction of the solution space. It is presented in Section 2.7.2, corresponding numerical experiments are discussed in Sections 4.1 and 4.2.

Chapter 3

Distributed Generic Component-Based Software Architecture to Simulate Probabilistic Models

The software system proposed and developed in this thesis to simulate probabilistic models is described in this chapter by its architectural structures, namely its software architecture. It was already introduced in [109, 108, 110]. The application of distributed generic software components indicates the main architectural concept. This concept enables one to combine the strengths of a generic programming and a distributed component-based approach and expresses itself in the following way. On the one hand a distributed generic component is located somewhere in a heterogeneous computational environment and, as a consequence, is flexibly bound at runtime. On the other hand such a component is accessed through a generic interface, that is to say an interface with generic types; as a result, a source code with generic type declarations is able to keep the abstractions of its own generic types, even if it instantiates a component. Furthermore, the generic idea on the implementation level can in fact be transferred to the runtime level. (Distributed generic software components and their mentioned properties are discussed in detail in Section 3.3.4.)

This chapter is structured as follows: frequently used programming languages in the field of scientific computing are outlined in Section 3.1. The general concept of a software architecture is clarified in Section 3.2. Software reuse in general, generic programming, distributed software components, and generic distributed software components in particular are focused in Section 3.3. The distributed generic component-based software architecture proposed in this thesis to simulate probabilis-

tic models is discussed in Section 3.4. Its characteristics are in the end summarised in Section 3.5.

3.1 Programming Languages in the Field of Scientific Computing

Programming languages can be divided into compiled, semi-compiled, or interpreted ones. The source code of a compiled language is transferred to a native code before execution; that allows fast processing. Code of a semi-compiled language is at first compiled into a binary code before execution and then interpreted to native instructions at runtime; that usually leads to slightly slower processing speed. Interpreted languages transfer the source code to native instructions at runtime usually resulting in slower processing. Prevalent compiled languages are `Fortran` [112], `C` [102], and `C++` [195]. `Java` [27] is a present semi-compiled programming language in the field of scientific computing, but far more popular outside of this field. `MATLAB` [79], `Octave` [174], and recently `Python` [113] mark the popular interpreted languages. Software for symbolic computations is not applied in this thesis, nevertheless well-established representatives should at least be named: `Maple` [179] and `Mathematica` [213]. Programming languages in the field of scientific computing are more comprehensively discussed in [204].

International standards have been established for each of the compiled languages. The consequence is a wide portability. Nowadays `C` and `C++` belong to the most popular languages, `Python` is steadily gaining popularity, while `Fortran` is falling back [1]. Nevertheless, `Fortran` is still used in scientific computing. This goes back to many basic, well-tried, and widely-used `Fortran` software packages for scientific computing providing high performance: these are, for instance, the linear algebra packages `BLAS` [117] and `LAPACK` [8], and the package `ARPACK` [119] for solving large eigenvalue problems. Corresponding application programming interfaces (APIs) for `C` are usually available. `Fortran`, `C`, and `C++` are known for their high performance: `Fortran` may exhibit slight advantages for special applications, but `C++` seems to maintain its performance over a large range of different applications [2]. More and more software packages in the field of scientific computing are implemented in `C/C++`, see for instance [65, 207, 75, 186]. The increasing value of `C++` can be explained by its high performance and its powerful high level mechanisms to abstract complexity. `C++` is based on the procedural programming language `C` and adds object-orientation; as a consequence, `C` code is mostly compatible

with C++ (see [196] for incompatibilities). A high level C++ binding to Fortran packages is discussed in [137]. A survey of C/C++ software packages and tools is presented in [48].

The Java programming language allows to write efficient high level code, which is frequently almost as fast as native code [2]. Recent packages for scientific computing are for instance proposed in [210, 10].

MATLAB provides a commercial higher level environment especially for scientific computations and corresponding visualisations with its own interpreted programming language and a comprehensive collection of packages; it can interface with C/C++, Fortran, and Java code. Octave is a comparable environment but under the terms of a free software license; MATLAB code is partly compatible to the Octave environment [174].

Python has recently become a promising and attractive programming language in scientific computing [113, 140, 168, 19, 46, 59]. The performance of Python code is far behind compiled code [2]. Strengths of Python are its high level structures and a rich variety of existing packages including also bindings to many different programming languages. As a consequence, a Python code may be used as a kind of administrative code for scientific computing; such a code strongly reduces the software complexity and interfaces native code to establish high performance. A Python based alternative for MATLAB, Maple, and Mathematica is proposed in [59].

3.2 Software Architecture

A software architecture describes the structure of a software system; the description includes the software units (elements) composing the software system, their interfaces, their mutual relations and interactions [17], and their physical distribution [15]. An interface is understood here to be specified by its syntactical and semantical characteristics; the latter may contain performance specifications as well. The physical distribution can be understood as the physical infrastructure — hardware specifications and network topology —, the distribution of software units within this infrastructure, and the used network protocols. The mentioned structures focus different views of the software system: one structure may for instance concentrate on a static view — the implementation —, while another one may concentrate on the runtime — the processes.

A software architecture reflects the details of software units concerning their interactions, and hides the details not affecting the interactions [17, 15]. In other words: it states the public (external) information and suppresses the private (internal) information of software units. In this sense a software unit is treated more like a black box (see also Section 3.3.1). Software units may be considered on different abstraction levels, from subsystems, components, and frameworks to packages and classes [15]. Software architecture facilitates the reuse of software units [68] namely on each abstraction level, see Section 3.3.

A software architecture may be based on distributed software components to realise a distributed software system, see Section 3.3.3. For more general details on software architecture refer to [17, 15]; different definitions of software architecture are discussed in [17].

3.3 Software Reuse

Regarding the software development an overall wish is to decrease costs by improving quality. As software systems have steadily increased in size and complexity, the call for software reuse has become louder. Generic programming and a decomposition of a software system into distributed software components are disjunct concepts to factor reusability into the software design and its development. While generic programming is a paradigm purely applied on the implementation level (and resolved at compile-time), distributed software components reveal their complete potential at runtime. The strengths of both concepts are combined in distributed generic software components.

Section 3.3.1 discusses software reuse in general, while Sections 3.3.2 and 3.3.3 centre on generic programming and distributed software components. Section 3.3.3 addresses distributed generic software components.

3.3.1 Software Reuse in General

Software reuse is not a recent idea. In 1969 Douglas McIlroy recommended to use reusable routines in software industry motivated by the paradigms in industrial electromechanics and mechanical engineering [135]. As a vision for the future he pro-

posed large collections of reusable routines in a black box sense categorised in families to allow comfortable choices, and to offer reliable and efficient software units. One proposition is to provide these routines for several abstraction levels — from general to tailored. This directly leads to automated code generation. Furthermore, he noted, that a user with a (domain-)specific interest may feel lost or may be annoyed by strong generality — which is sometimes disregarded. In many respects these ideas are still up-to-date.

A common technique is the *opportunistic reuse* [187], in which external code is copied and pasted into the internal one. Obviously, this approach causes trouble in maintenance, as bugs must be fixed over all of the technically redundant copies. A promising approach is the *systematic reuse*, in which code reuse is organised by sophisticated techniques and analysed already before and/or during the development process, see [187, 64].

So far the term *software reuse* has not been defined. Software reuse means the reuse of software or of the knowledge to produce a specific software [64]. As a consequence, software reuse has many levels of granularity, for example the reuse of ideas, concepts and patterns, algorithms, classifications, interfaces, implementations, libraries, subsystems, or whole software systems. For any of these items the term *reusable asset* may be used; the *reusability* of such an asset characterises its probability to be reused [64]. The more these assets are designed with respect to reuse, the higher the ability to reuse them in other contexts or systems. Of course an item which uses a reusable asset is preferentially a reusable asset too, because also when the item is not reused until now it may be reasonable to reuse it in the future.

Reusable code assets may be distinguished into *white box* and *black box* assets, see e.g. [176]. The code of white box assets is accessible and may be modified to satisfy the needs. Apparently, for modification one needs to be familiar with the source code. For in-house assets this may be ensured, but for third-party assets this might not be the case. Obviously an adequate documentation is essential and a long period of vocational adjustment needs to be scheduled. A black box asset hides its implementation and captures all required knowledge for its invocation in its interface. It is noted that signatures (of methods) alone do not make up the interface. Beside its syntax, the semantics — what a function means and in which order it has to be called — needs to be clear. The code of a black box asset may be accessible, but its modification is not intended, and knowledge about the internals is not necessary for its invocation. The flexibility of a black box asset depends on its abstraction or its adjustable parameters/switches. To realise assets of high flexibility a designer or developer is required to be highly experienced and to have a broad knowledge [187].

Black box assets are usually developed by experts in the proper topic. For a concrete application of a black box asset the parameters/switches need to be specified.

Unfortunately, software reuse is complicated and may also become risky: one can imagine a reused third-party library which after a while is not maintained anymore or restricts the software — which uses the library — to be ported to another machine. As a consequence, the asset to be reused needs to satisfy many (non-functional) properties which are mentioned in the following (see also [187]). It should be *reliable*, meaning that it works as it is specified. It should be *maintainable* so that e.g. run- or compile-time problems can be solved as desired in an easy manner. From a special asset complexity upwards it should be *maintained* by a third-party community — which needs to be also reliable. The asset should be *extensible* so that desired additional features can be realised, and it should be *portable* to allow a usage in a different environment. Additionally, it needs to be *flexible* so that it is reusable at all. When it exhibits a high flexibility or the desired context is close to the original one of the asset an embedding may be possible directly or with modifications at acceptable costs. Last but not least an asset has to be well-documented, otherwise the amount of vocational adjustment may become too high and the intrinsic sense of software reuse to reduce development time is lost. Haphazard or no documentation at all may reflect weakness in the whole software development and maintenance processes and is not professional. The better an asset satisfies the previously mentioned properties the more had been invested for this purpose. The properties may be considered with different priorities in the design and development process of an asset, or when choosing an asset for reuse.

Software reuse can be complicated, risky, and time-consuming, but it is a promising possibility to handle the increase of software size and complexity and improves software quality at the same time. For instance, concepts of modern programming languages have already made a contribution to software reuse. The object-oriented approach allows a more intuitive perspective on programming. Classifications of objects and their properties provide a real-world point of view. Inheritance captures similarities in structure or behaviour. Polymorphism allows data/operators to be of/work on different types. Exception handling allows to classify and react on abnormal program flow. Generic data types allow to implement an algorithm for many different specific data types without the demand of an inheritance relation between them, see Section 3.3.2. These concepts enable a better human readability of source code by simultaneously reducing the lines of code compared to vintage programming languages. Another idea is to separate a software system into preferably many reusable software components. An architectural composition of software components may notice a software component to be a black box asset, see Sections 3.2 and 3.3.3.

3.3.2 Generic Programming

There is no single definition for *generic programming* [144, 45, 180]; different definitions are discussed in [45, 180] and vary from broader to more precise notions. It can be understood as a paradigm in which an algorithm is abstracted into its data types to gain a reusable algorithm. This leads to an abstraction of the data types and indicates the abstracted algorithm to be independent of specific data types. An abstracted data type is known as a *generic type* or a kind of a *generic parameter*; for formal reasons the abstracted algorithm and its implementation may also be called generic. The generic types act like placeholders with specific constraints. The constraints are prescribed by the generic algorithm and comprise the syntactical and semantical use of the generic types inside the algorithm.

A corresponding generic implementation is an image of the generic algorithm written in a programming language. Specific data types are realisations/specialisations for the generic types if they fulfil the constraints. Accordingly, the generic implementation can be used for all the specific data types which satisfy the constraints. This ability is called *parametric polymorphism* [45].

In contrast to parametric polymorphism, the (subtype) polymorphism in object-oriented programming (OOP) is induced by a family relationship between data types. However, implementations like container classes should match a wide variety of specific data types which do not exhibit a natural family relationship. That is exactly the fundament for parametric polymorphism [49]. A generic code is specialised by specific data types to allow its compilation. As a consequence, a member function call is plain (non-virtual) at runtime. In contrast, a virtual member function call (by a subtype polymorphic object) is resolved at runtime usually by a virtual function table; this costs twice as much as a plain call [45]. That is significant when functions of small content need to be called many times.

Generic programming obviously increases the reusability of a code. A very popular generic library is the `C++ Standard Template Library` (`STL`) [194]. `C++` supports generic types by the principle of *templates* [195]. A template class enables one to implement an algorithm on generic types, the so-called *template parameters*. The constraints of these template parameters are called their *concept* [78]. A `C++` template class `Vec` for a vector representation is for instance explained in Listing 3.1 with template parameter `real`. `real` may be specialised by a fundamental type for a real number, for example `double` or `float`. Templates are used in many software packages in the field of scientific computing, see for instance [207, 75, 186]. However, other programming languages enable generic programming as well [67].

```
1 template<class real> class Vec { /*...*/ };
```

Listing 3.1: *A C++ template class `Vec` for a vector representation.*

3.3.3 Distributed Software Components

The definition of a software component varies from broad to specific [45, 197, 97, 106], comparisons can be found in [114, 93]. A widely-used definition is published in [197]. There, a component can be understood as being necessarily stateless [93]. However, a state may be required for performance reasons in scientific computing. The understanding of a software component in this thesis and associated terms and definitions are given in the next paragraphs and are partly summarised in framed text blocks according to their relevance.

Here, a software component is in the first place understood as a black box asset (see Section 3.3.1). (Other works may also involve white box software components.) It is a software unit separated in its interface — the *component interface* — and its implementation or binary code representation — the *component implementation*. As a consequence of the black box appearance the software component is exclusively specified by its component interface. A component is coupled with other components (or code in general) to compose a software system. The coupling is loose [164, 14] due to the exclusivity of the component interface. A software component is reusable, as it may be coupled many times inside the same or various software systems. The next framed text block summarises a definition for a software component.

> A *software component* is a reusable software unit exclusively specified by its interface.

A number of further terms and definitions concerning software components and their environment will be stated as well, corresponding literature references are for instance [114, 93]. The general representation, composition, and environment of software components are described by a component-based software architecture. Its *software component model* defines the technical construction, assembly, and usage of software components. A corresponding *(component-based) framework* implements a software component model and can be used to develop applied (concrete or domain-specific) component-based software architectures.

The reusability of a software component does not imply an interchangeability of components. Nevertheless it is advantageous to allow an interchange of components. Appropriate reasons for an interchange are, for instance, the following ones: a component may need to be interchanged due to upcoming licence conflicts, or simply because another component implementation is of higher performance. Anyhow, the question arises when a component can be exchanged for another component. To not cause formal conflicts, the view onto the component interface needs to be intensified. The exclusivity of the component interface means its completeness: the component interface defines all specifications of the software component. These are the necessary ones for the usage and the reasonable ones for the promotion of the software component. In other words, the specifications can be separated into functional and non-functional ones. The functional specifications answer the questions about the actual task of an appropriate software component and how it should be used; these specifications are described both syntactically and semantically. The non-functional specifications answer the question why exactly the provided component implementation should be used; these specifications are, so to speak, special features of the component implementation and are, for instance, related to the performance or the robustness. The non-functional specifications are clearly the ones which may motivate to interchange a component. Conversely, the functional specifications are the ones which need to match for an interchange. A distinction between functional and non-functional specifications is obviously required to be able to speak about interchangeability of components in a consistent manner. This awareness results in the following definitions.

The component interface of an interchangeable software component is formally separated into functional and non-functional specifications.

A software component can be interchanged with another software component when the other component satisfies the functional specifications required in the appropriate scenario.

The architectural composition of software components usually emphasises the functional specifications and neglects the non-functional ones. The non-functional specifications are an essential argument for the binding of concrete software components at runtime.

Scientific computing may make additional demands on both the software component itself and the framework to realise a concrete component-based software system.

Software components may be distributed to gain from the resources of many computing machines. Access to a maybe remotely located software component postulates a late binding (to this component), namely a binding at runtime. As a consequence, the knowledge about the location is essential only at runtime. (By contrast, the component interface is also required at compile-time.) The late binding of distributed software components enables an easy interchangeability of such components at runtime.

A software component (for instance from a third-party) may be only available in binary form on a dedicated machine. However, a software component may be also independently distributable (for flexibility reasons). In that case the component should be available by its source code to allow its (re-)compilation on the executing machine for performance reasons. In this regard, a software component is assumed to be closed in the sense that code modifications of the component itself are not required for a distribution. (A software component only available in binary form is by default closed.) All these demands and ideas will be summarised by the further definition:

> A *distributed software component* is a lately and locally or remotely bound, closed, and interchangeable software component.

The demands on a corresponding distributed component-based framework can be named as follows: a framework should provide interoperability between preferably many programming languages (including the popular ones of scientific computing, see Section 3.1), and between different operating systems (and maybe hardware). Last but not least the pure communication overhead induced by the bindings should be as small as possible to permit high performance computations. Certainly the underlying software component model may already restrict the framework regarding these features.

Appropriate component-based architectures are for instance the *Common Object Request Broker Architecture* (*CORBA*) [84] and the *Common Component Architecture* (*CCA*) [220, 9], for which a number of frameworks exist implementing their corresponding software component model. The CCA is frequently included for component-based approaches in scientific computing, see for example [118, 92]. The *Component Template Library* (*CTL*) [151] is an independent framework to realise distributed component-based software systems. A corresponding software architecture and its software component model can be extracted from the CTL's communication protocol [33]. The CTL is a C++ template library, but also binds native binaries written in `C`, `Pascal`, and `Fortran`; a Java implementation of the communication

protocol [33], Python and MATLAB bindings [34] are also provided. More about the CTL can be found in Section 3.4.1. Each architecture specifies its own (usually programming language independent) *interface description language* (*IDL*) to define component interfaces.

Here a distributed software component is instantiated like an object in object-oriented programming. (This is comparable to the concept of the *remote method invocation* (*RMI*) introduced by the programming language Java [214].) In this context, a local object binds an instance of a distributed software component for a possibly remote interaction at runtime. The local object is a one-to-one reflection of the component interface. This should be exemplified: rCI is meant to be such a local instance through which a distributed software component can be instantiated and accessed. The extension "CI" is an abbreviation for component interface; it is attached to accentuate that the component is accessed through its interface, and that the interface is reflected by the local object. An invocation of an available method provide is presented in Listing 3.2 using a C++ notation. It is obvious that the actual complexity of the possibly remote access is completely hidden.

```
1 float f = rCI.provide();
```

Listing 3.2: *Local instance to access a remote software component.*

3.3.4 Distributed Generic Software Components

The strength of a distributed software component is its interchangeability at runtime; the strength of generic programming is to implement algorithms on generic types so that the implementation can be directly reused for many concrete data types between which a family relationship is not necessarily recognisable. Distributed generic software components connect both conceptions and possess both mentioned strengths. The connector is induced by *generic component interfaces*. A generic component interface is indicated by generic type declarations. Such a parametric polymorphism for distributed software components [40, 162, 161, 160] is relatively recent and rarely addressed. Software support is provided by *GIDL* (*generic interface definition language*) [162, 40] — a CORBA-IDL extension — and the already introduced CTL. In the following the need for generic component interfaces is demonstrated. The definition of a distributed generic software component is summarised in a framed text block at the end of this section.

Generic programming has its strength on the implementation level, while distributed

software components emphasise their strength at runtime. The generic support vanishes at compile-time and is actually not available anymore at runtime. Nevertheless, a conflict is induced when distributed (non-generic) software components are instantiated in a generic code. This is illustrated by the C++ template class in Listing 3.3, in which a distributed (non-generic) software component is instantiated and accessed through its method `provide`.

```
1 template<class real> class Algorithm {
2     /*...*/
3     real perform( /*...*/ ) {
4         /*...*/
5         RCI rCI( /*...*/ );
6         real r = rCI.provide();
7         /*...*/
8     }
9 };
```

Listing 3.3: *A C++ template class invoking a local instance to access a remote software component.*

It is assumed that the method `provide` returns a `float`. Then the variable `r` can at most keep the precision of a `float`. When `r` has an impact on the response of method `perform` the response is as well at the most of `float` precision independent on the specialisation for the template parameter `real`. Consequently, the template class `Algorithm` lost the abstraction of its generic type `real`. There is only one possibility to rescue the generic implementation: the object `rCI` needs to be an instance of a generic class with a generic type like `real`, see Listing 3.4. Then, method `provide` can return the proper type.

```
1 template<class real> class Algorithm {
2     /*...*/
3     real perform( /*...*/ ) {
4         /*...*/
5         RCI<real> rCI( /*...*/ );
6         real r = rCI.provide();
7         /*...*/
8     }
9 };
```

Listing 3.4: *A C++ template class invoking a local instance to access a remote generic software component.*

This postulates the interface to be generic in terms of types. In summary, the following definition can be formulated:

A *distributed generic software component* is a distributed software component explicitly specified by its **generic** interface. The interface is called generic, as it contains generic type declarations.

The question may arise how a generic component interface influences the potential of a software component at runtime. While compiling the source code of a software component the generic types of the interface are of course specialised. Here the binary outcome of a specialisation and compilation is called *specialised distributed software component*. It may be represented by an executable. Each other desired specialisation for the generic types would lead to another specialised distributed software component in form of an executable. These executables can be distinguished by different filenames. As a distributed software component is lately bound it can be assumed that its location identifier also contains the filename of the executable. Then, software components can be consistently interchanged at runtime (by interchanging the location identifiers) to change the data type specialisations. This maintains the generic idea at runtime.

3.4 Distributed Generic Component-Based Software Architecture to Simulate Probabilistic Models

Many concerns in the field of probabilistic models and their simulations can be separated. A concern is implemented here by a single (interchangeable) distributed generic software component or a composition of such components (depending on the abstraction level of the concern). The proposed software architecture [109, 108, 110] describes all these components and their relations with regards to each concern. An essential concern within the simulation of a probabilistic model is the construction (and potentially the simulation) of deterministic models. This is implemented by a proper component, namely a deterministic simulator component. It reflects general conceptions in its component interface and interfaces an individual simulation code (probably from a third-party) in its component implementation. As a result the proposed software architecture is invariant to the application-specific semantics of the probabilistic model and individual simulation codes for deterministic models. Different software frameworks are developed for different (kinds of) numerical schemes to simulate a probabilistic model. A framework is composed by the appropriate components. In this context each of the VLR-SR1U, VLR-OPT, and RBSSE algorithms is

realised through a flexibly configurable generic code. The software components and frameworks are mostly implemented in C++, using the Component Template Library (CTL) among others.

The application of the CTL is addressed in Section 3.4.1. Conventions and formalisms for the architecture and its graphical illustration within this thesis are focused in Section 3.4.2. The distributed generic software components and software frameworks developed for certain concerns are discussed in Section 3.4.3. The general conceptions of a deterministic simulator component as well as corresponding individual component implementations for different application-specific semantics are presented in Section 3.4.4. The generic implementations of the VLR-SR1U, VLR-OPT, and RBSSE algorithms are demonstrated in Section 3.4.5. The proposed software architecture to simulate probabilistic models is summarised in Section 3.5.

3.4.1 Application of the Component Template Library

The Component Template Library (CTL) — already presented in Section 3.3.3 — is the framework used to realise the distributed component-based software architecture proposed in this thesis to simulate probabilistic models. Essential reasons for its application are quite simple: the CTL is an in-house product and supports generic types, it enables one to develop distributed generic software components, and also provides a good performance for scientific computations.

The CTL is a C++ template library and ports many C++ specific features to software components, which are inherently innovations for a component-based architecture. These features are for instance the possibilities to overload operators (to allow mathematical notations on the implementation level), to declare inheritances between component interfaces, and — as already mentioned — to declare generic types. A component interface (CI) is directly written in C++ and linked to standard C++ headers or implementations; a CI is nothing else than a C++ class, and as a consequence the concept of RMI is directly induced. This enables one to implement plain C++ code without a dependency to the CTL but operating on objects potentially bound to distributed software components; at this, the potential distribution is delivered by passing inheriting objects or instances of template parameter realisations, see Section 3.4.5 for illustration. The CTL provides diverse bindings for a software component at runtime: a component can be remotely bound as an executable, or it can be locally bound as an executable or for performance reasons as a thread or a library.

Its support concerning the interoperability between programming languages is out-

lined in Section 3.3.3. For instance FORTRAN77 does not have object-oriented constructs. Nevertheless the CTL understands how to access native code compiled from FORTRAN77 sources. However, CIs compatible to FORTRAN77 are restricted regarding applicable data types.

The CTL predefines an abstract type declaration array<T> for a container, which can be optionally used inside a CI. Type T is a generic or specific data type; array<T> is specified through the data serialisation for the (potential interprocess) communication at runtime. The vector containers std::vector<T> of the STL (see Section 3.1) and ctl::vector<T> of the CTL are, for example, specific representations for array<T>, that means their serialisations match the specifications of array<T>. As a consequence, a code — the *caller* — instantiating a CI may use one representation, while the code — the *callee* — implementing the CI (namely the software component itself) uses another representation. The serialisation mechanism enables one to introduce own abstract type declarations, which may reflect quite complex data types. Section 3.4.4 applies the abstract type declaration array<T>.

3.4.2 Conventions and Formalisms

In this section conventions, formalisms, and annotations for the proposed distributed generic component-based software architecture and its graphical illustration are mentioned which are henceforth valid. The software components and frameworks are mostly implemented in C++. On the one hand, C++ is an accepted programming language in the field of scientific computing (see Section 3.1), on the other hand it is the most direct link to the CTL (see Section 3.4.1).

A software component is illustrated through abstract representatives, namely a separated and a non-separated representative. The illustrations are shown in Figure 3.1. The separated representative considers both the CI and the component implementation of a software component separately. The non-separated representative merges the CI and the component implementation. It takes the name of the corresponding CI. The name of a CI ends with "CI". The *use* relation indicates that a component, or in general a code, uses another component. When one component uses another component its component implementation uses the CI of the other component. However, details about the component implementation are non-essential for an architectural description, and consequently the non-functional specifications of the CI are as well non-essential (see Section 3.3.3).

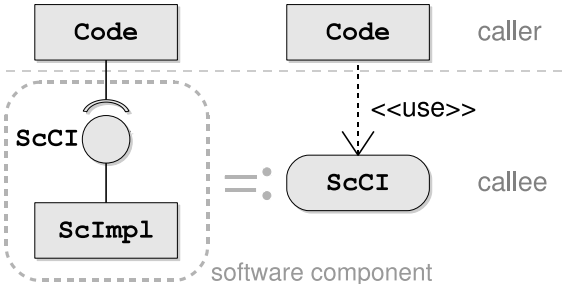


Figure 3.1: *Representatives for a software component: a code interfaces to a software component through the component interface (CI); the separated representative of a software component is shown on the left side, the non-separated representative is shown on the right. The non-separated representative merges the CI* `ScCI` *and the component implementation* `ScImpl` *of the separated representative and keeps the name of the CI.*

As already mentioned in Section 3.3.3, a software component is formally treated as an object in object-oriented programming. The CI is reflected by a class declaration. A software component is locally instantiated and accessed through that class declaration and may be executed remotely. The class representation of CIs enables the definition of inheritance relations between them. At this, a CI can inherit the method declarations from another CI. The inheritance relation carries a formal order to CIs and should increase the overall understanding and clarity.

In summary, two different kinds of relations are considered: the use relation between software components, and the inheritance relation between CIs. All stated specifications for the representation and handling of software components are conformable to the CTL. The general elements of architectural diagrams are summarised and explained in Figure 3.2. The corresponding illustrations are partially motivated by the *Unified Modeling Language UML* [163].

Here, a general convention prescribes more fundamental data types inside a CI, that means complex data types are not allowed both for the arguments of constructor declarations and for the arguments and return types of method declarations. This convention is established to enable interoperability to programming languages which do not exhibit structured data types (see also Section 3.4.1). However, these more fundamental data types are abstracted to generic types. The generic types are `real` and `integer` for a real number and an integer number; for instance valid data type real-

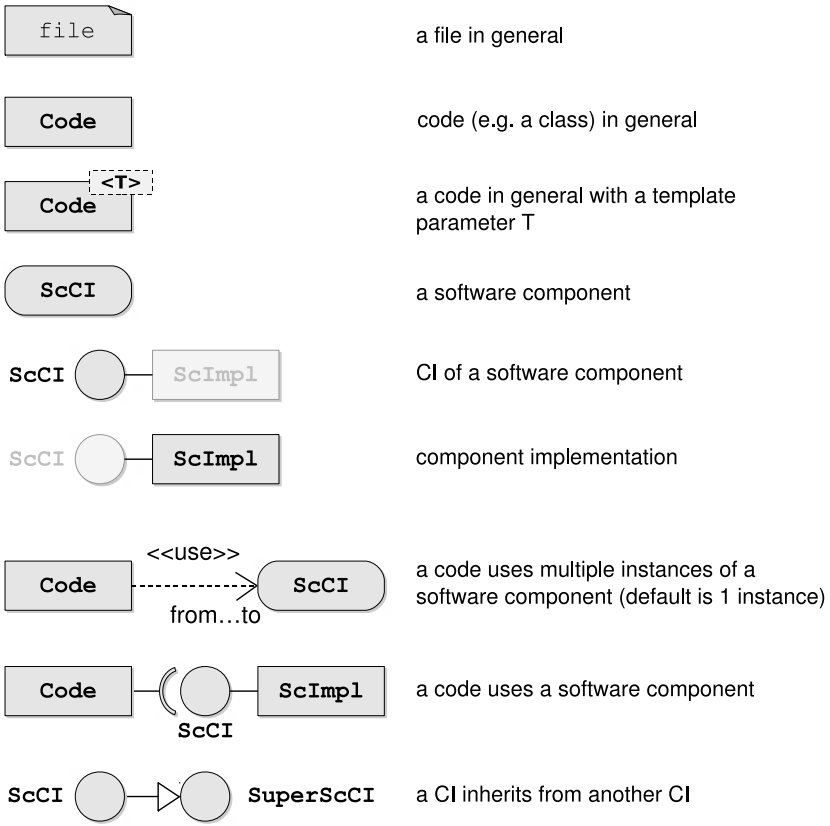


Figure 3.2: *The general elements of architectural diagrams used in this thesis: the identifier for template parameters can also be attached to software components, component interfaces (CIs), or component implementations.*

isations for `real` are `float` or `double`, valid data type realisations for `integer` may be `int` or `unsigned int`. The abstract type declarations `string` for a string and `array<T>` for a vector container — predefined by the CTL (see Section 3.4.1) — are as well applied. `array<T>` is used for each of the previously named generic types. Operator overloading — supported by the CTL — is not applied for the same reason of language interoperability.

When a CI is presented in its detailed declaration, it is done in CTL conformable

C++ syntax. Code in general is presented in C++ notation, generic programming is realised by C++ templates.

3.4.3 Concerns and Their Distributed Generic Software Components

Many different concerns can be identified in the simulation of probabilistic models. These concerns are driven by the demands of the preprocessing step, the simulation (or solving) step, and the post-processing step, but also by the demands of the different kinds of numerical schemes applied for the proper simulation. Some concerns and their corresponding distributed generic software components are exemplified in this section. (Software components are considered through their abstract representatives, a more precise insight into a CI is provided in Section 3.4.4.)

The concerns of a deterministic simulation and a potential geometrical discretisation are discussed in Section 3.4.3.1; the concern to compute a truncated KLE is discussed in Section 3.4.3.2. Developed frameworks using a number of software components are presented in Section 3.4.3.3. Section 3.4.3.4 summarises the overall content.

3.4.3.1 The Deterministic Simulation and the Geometrical Discretisation

The numerical schemes to simulate a probabilistic model may be subject to quite different concepts. Nevertheless, the probabilistic model needs to be usually either constructed or simulated for samples (realisations) of its intrinsic random variables. A sample of the random variables carries the probabilistic model over to a deterministic one. A deterministic model and its construction or simulation obviously depicts one concern. The concern is analysed for the different numerical schemes and reflected by a set of software components, each of them referred to as a *deterministic simulator component*.

Furthermore, a probabilistic model may depend on geometrical variables besides stochastic ones. If so, these variables are usually discretised. The discretisations of the geometrical domain for the uncertain parameter and the model solution do not necessarily need to match. Thus, a discretisation may be applied in the preprocessing step detached from the simulation (for instance to model an uncertain parameter), but it may also be applied in the simulation step (and accessed in a post-processing step to

gain a more readable representation of results). The discretisation of the geometrical domain is the second concern reflected in a set of software components, each of them referred to as a *discretisation component*. When a geometrical domain exists it is an integral part of the deterministic model and, consequently, the corresponding model construction and simulation. This dependency is as well reflected by the software components of both concerns through inheritance relations between the component interfaces (CIs).

Different numerical schemes may make different demands on the mathematical representation of the (discretised) probabilistic model. These demands are transferred to a proper deterministic simulator component for each different kind of numerical schemes. Direct integration, projection, or collocation schemes are based on sampling techniques. An associated deterministic model can be mathematically understood as a simple black box, which is evaluated (simulated) for a sample of the uncertain parameter; it does not need to reveal its intrinsic mathematical representation, and only a minimum of interactions is required for a simulation. Through these interactions the deterministic model is set and evaluated for a sample of the uncertain parameter, and the corresponding deterministic solution is provided. The deterministic simulator component matching these demands is referred to as the `NISimuCI`. In contrast, numerical schemes based on the SGM already have a detailed understanding of the mathematical representation of the discretised probabilistic model and ask exactly for that representation: the discretisation of the geometrical domain is done by finite elements, the discretisation of the stochastic domain is done by a truncated PCE; as a result the simulation of the probabilistic model solves a system with left- and right-hand side declarations. This mathematical representation is reflected in the deterministic model. Therefore, the associated deterministic model can be mathematically understood as a white box. It enables, for instance, the evaluation of the left-hand side operator corresponding to the mathematical representation of a deterministic model. The deterministic simulator component matching these demands for the case of a linear model is referred to as the `FELinSimuCI`.

The use of the term “black box” in the previous passage is mathematically motivated. It should not be mixed up with the term “black box asset” from software engineering, see Section 3.3.1. While the deterministic models may mathematically reveal black box or white box appearances inside the mentioned numerical schemes, their associated deterministic simulator components are by default black box assets. As well, in the literature the terms “non-intrusive” and “intrusive” are popular in this context. These terms are usually motivated by a preceding detailed understanding of the mathematical representation in the sense of a white box and focus the demands on a third-party simulation code to enable its application for different kinds of numerical schemes, refer to Section 3.4.4 for further details. Nevertheless, it is already

stated that the term "NI" in the naming of NISimuCI stands for "non-intrusive".

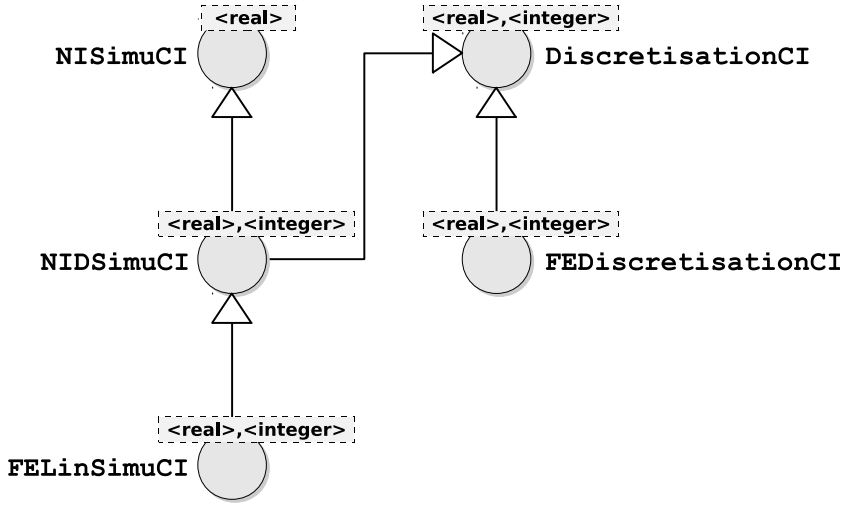


Figure 3.3: *The inheritance relations between the generic CIs associated to the distributed generic software components for a discretisation of the geometrical domain and a handling of deterministic models (their construction and simulation).*

The previously discussed concerns are now completely resolved by software components. Their CIs and associated inheritance relations are presented in Figure 3.3. The software component DiscretisationCI provides information on the discretisation of the geometrical domain, namely its mesh defined by the nodes and the connectivity. FEDiscretisationCI represents a finite element based discretisation and extends DiscretisationCI by a method to construct a mass matrix; by this means it comprises the assembly parts of the geometrical discretisation located in the computation of a KL representation, see Section 3.4.3.2 for more details. NIDSimuCI extends the already introduced NISimuCI to supply information about the discretisation of the geometrical domain (helpful for a post-processing step). For this purpose it additionally inherits from DiscretisationCI. The inheritance relations enable one to apply each of the three deterministic simulator components in conjunction with a direct integration, projection, or collocation scheme.

The generic types real and integer of component DiscretisationCI specify the type of a single nodal coordinate and the type of a single nodal id. The generic

types of the other components have similar meanings. More details about them are presented in Section 3.4.4.

3.4.3.2 The Computation of a Truncated non-Gaussian KLE

The concern assigned in this section is the computation of a truncated KL representation of a non-Gaussian random field geometrically discretised by finite elements. This concern can be resolved by a number of software components connected through the use relation. Some components satisfy a more general subject and may be also interfaced beyond the proposed software architecture.

The truncated KL representation based on a finite element (FE) discretisation of the geometrical domain is formulated by Eq. (2.15) under the consideration of Eqs. (2.16) and (2.17) in Section 2.1.1.2. The appropriate computation requires to solve the general eigenvalue problem described by Eq. (2.16) involving a covariance matrix and a mass matrix. The covariance matrix reflects the covariance function of the non-Gaussian random field evaluated on the defined FE mesh. Furthermore, the PC coefficients of the expanded random variables located in the KL representation need to be determined; the latter is formulated by Eq. (2.17).

Figure 3.4 illustrates the generic software components connected through their use relation to compute a FE-discretised truncated non-Gaussian KL representation. The software component `KleCI` uses `CovCI` and `MIGenCI`. `CovCI` constructs the assembly parts of the general eigenvalue problem and solves it. In this context, it obtains the information about the FE mesh and the appropriate mass matrix from `FEDiscretisationCI`. The software component `CovFCI` implements the covariance function; `CovCI` uses `CovFCI` to evaluate the covariance function on the FE mesh to construct the covariance matrix. `EvSolverCI` solves the constructed general eigenvalue problem. The software component `MIGenCI` generates multi-indices indicating the stochastic polynomials of the truncated PCEs which discretise the random variables located in the KL representation.

The provided component implementation of `EvSolverCI` interfaces the package `ARPACK` [119] (or more precisely `ARPACK++`, a `C++` wrapper of `ARPACK`) and uses ARPACK's *reverse communication interface* (*RCI*). The RCI provides a mechanism to embed own matrix representations. Among others, the RCI asks for a linear solver. The latter is provided by the software component `LinSolverCI`. However, another component implementation of `EvSolverCI` may not ask for a linear solver, so that `LinSolverCI` is then not interfaced. `EvSolverCI` and `LinSolverCI` are software components which may also be used beyond the proposed software architecture.

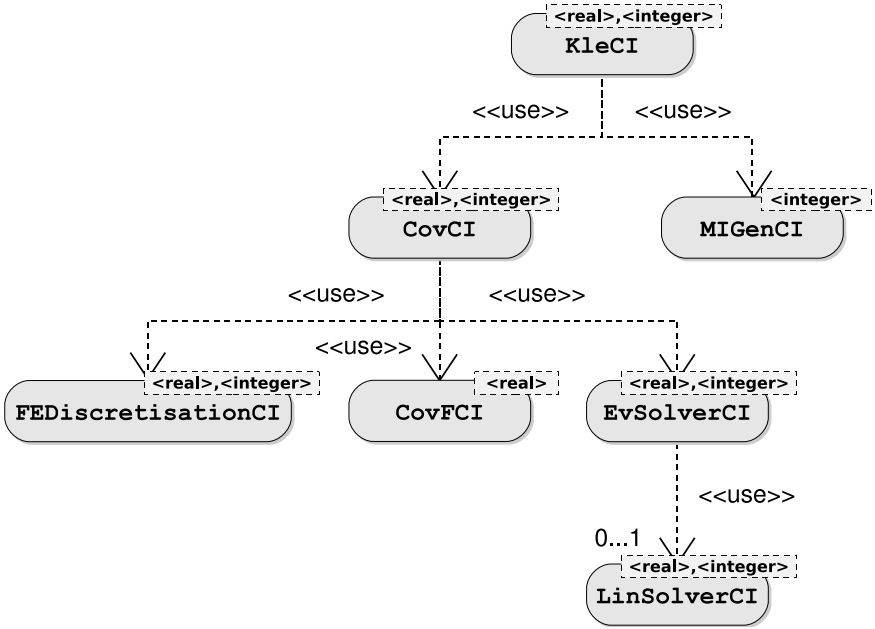


Figure 3.4: *Generic software components connected through the use relation to compute a truncated non-Gaussian KL representation geometrically discretised by finite elements.*

3.4.3.3 Frameworks

Several frameworks are implemented to simulate a probabilistic model [109, 108, 110]. Each of them realises a different numerical scheme or a different kind of numerical schemes. Figure 3.5 compares the frameworks, which implement direct integration, stochastic collocation, and SGM-based schemes.

The frameworks for direct integration and stochastic collocation simply interface the deterministic simulator component `NISimuCI`, while the SGM-based framework interfaces the deterministic simulator component `FELinSimuCI`, for more details on the different kinds of deterministic simulator components see Sections 3.4.3.1 and 3.4.4. The stochastic collocation and SGM-based schemes represent the solution of the probabilistic model in a truncated PCE. For this the corresponding

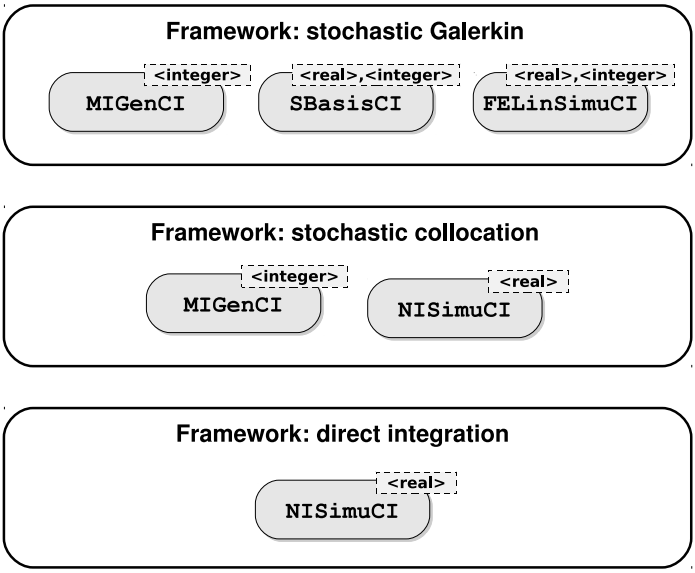


Figure 3.5: *Comparison of different frameworks to simulate a probabilistic model in respect of their involved generic software components.*

frameworks use the software component `MIGenCI`; it is a generator for multi-indices indicating the stochastic basis polynomials. The SGM-based framework additionally uses a software component `SBasisCI`; this component constructs the parts of the operator introduced by the SGM through a stochastic discretisation, namely the matrices $\mathbf{\Delta}_\gamma$ in Eq. (2.61) of Section 2.6.2 (or, analogously, the matrices $\mathbf{\Delta}_i$ in Eq. (2.64)). The SGM-based framework implements the algorithms VLR-SR1U, VLR-OPT, and RBSSE. These implementations are focused on in Section 3.4.5.

3.4.3.4 Conclusion

Probabilistic models and their simulation possess various concerns on different abstraction levels. In this manner, a concern may include a sub-concern, which is eventually shared by other concerns. Addressed concerns are for instance the modelling of an uncertain parameter through a KL representation, and the construction or simulation of deterministic models. The first, and potentially the second concern, are

based on a discretisation of the geometrical domain which identifies a shared subconcern.

The computation of the KL representation is divided into several tasks, and each of them is executed by a distributed generic software component. Some of these components are quite obviously reusable beyond the field of probabilistic models and their simulation, see Section 3.4.3.2. A parameterised deterministic model and its simulation is provided by a distributed generic software component, namely a deterministic simulator component. The different numerical schemes may have different demands for the deterministic models. These different demands are reflected by different component interfaces (CIs) for deterministic simulator components. The CIs exhibit inheritance relations which express an order from low to high demands (and vice versa) of different numerical schemes, see Section 3.4.3.1. The discretisation of the geometrical domain is represented by other distributed generic software components. Such a discretisation component is once used in the computation of the KL representation, and once bound in the mentioned inheritance relations, see Section 3.4.3.1.

Frameworks are implemented for the direct integration, a stochastic collocation, or an SGM-based simulation. Each framework uses a number of distributed generic software components, see Section 3.4.3.3.

3.4.4 Deterministic Simulator Components

The simulation of a probabilistic model may implicate deterministic models associated with samples of the uncertain parameter. Probabilistic models may be subject to quite different application-specific semantics, and consequently this applies for the deterministic models as well. Here, for instance, the subjects are groundwater flow, structural mechanics, and the preliminary design of an aircraft.

Third-party simulation codes for deterministic models are usually available for each of the representative subjects. An appropriate code may be built on expert knowledge and developed in many man-hours. It may be robust, efficient, rich in features, maintained, reliable, and hence recommended. So, it is obvious or maybe indispensable for reasons of time to interface a proper third-party simulation code, rather than to develop a code from scratch. However, such third-party simulation codes are quite individual in their structure, handling, and workflow.

The software architecture proposed in this thesis is **invariant** to the application-

specific semantics and structurally individual simulation codes for deterministic models, except at runtime. This is traced back to the introduction of a deterministic simulator component. It represents or simulates a deterministic model inside the software architecture and exhibits the same invariance. The invariance is obtained by the corresponding component interface (CI). The CI reflects the general conceptions of a deterministic simulator, while the component implementation deals with the individual conceptions. The component implementation implements the CI and interfaces the individual simulation code (of course not necessarily from a third-party) for deterministic models. For this purpose it translates between general and individual conceptions. In other words, the software architecture allows to replace the deterministic simulator component — and, consequently, the application-specific semantics and the interfaced simulation code — at runtime.

Here different kinds of numerical schemes are discussed to simulate a probabilistic model. As the case may be they make more or less demands on the mathematical description of the probabilistic model, which are automatically carried over to the deterministic models. The different demands are reflected in different kinds of deterministic simulator components represented by different CIs. The corresponding CIs `NISimuCI`, `NIDSimuCI`, and `FELinSimuCI` and their inheritance relations are already introduced in Section 3.4.3.1. (The names of the CIs are alternatively used for the corresponding deterministic simulator components.)

This section is structured as follows: the CIs of the deterministic simulator components are discussed in Section 3.4.4.1 in respect of syntax and semantics. The separation of general and individual conceptions is closely examined. The third-party codes interfaced here to simulate deterministic models of different application-specific semantics are focused in Section 3.4.4.2. A concrete component implementation of a deterministic simulator component for a preliminary aircraft design is exemplified in Section 3.4.4.3, and the translation between general and individual conceptions is demonstrated. Section 3.4.4.4 summarises the overall content.

3.4.4.1 Component Interfaces

The deterministic model — represented by a deterministic simulator component in the proposed software architecture — can be configured through a parameter of the component. Accordingly, the deterministic simulator component is a parameterised simulator. The parameter is a placeholder for a sample of the uncertain parameter and can be set at the latest at runtime. Here, for instance, the parameter indicates the hydraulic conductivity in a groundwater flow, the material of a laminated composite

structure, or physical quantities of an aircraft. It is defined as an input argument of constructors and methods in the different CIs of the deterministic simulator components.

These CIs are `NISimuCI`, `NIDSimuCI`, and `FELinSimuCI`. Inheritance relations are defined between them, see Section 3.4.3.1. In this section the CIs are exemplified in their syntax and semantics with an emphasis on the simulation step. `NIDSimuCI` simply extends `NISimuCI` by methods to obtain the nodes and the connectivity of the discretised geometrical domain. These methods are reasonable for the post-processing step, but not for the simulation step. As a consequence, `NISimuCI` and `FELinSimuCI` are considered in the following while further explanations about `NIDSimuCI` are omitted.

A CI is written by C++ preprocessor directives and its declaration is oriented towards the declaration of a C++ class. Input and output arguments of constructors and methods which are declared inside the presented CIs can be distinguished as follows: an argument with a prefixed `const` cannot be modified and consequently represents an input argument; the other arguments are output arguments and understood to be called by reference (thus one can notionally put the C++ reference operator `&` in front of the argument type). As a consequence, a method may provide its output through arguments and not through the return statement (then `void` is returned). The call by reference is emulated by the CTL for remote invocations. The keyword `const` may also be declared after the closing bracket of the argument list of a method. That declaration indicates that the method does not modify class members.

`NISimuCI` declares the methods to set the parameter, to solve the appropriate deterministic model, and to obtain the outcome. These methods (or interactions) only ask the deterministic simulation code for functionality which usually the code already provides through its user interface. Therefore, the methods are denoted as non-intrusive ones. This denotation is also reflected by the "`NI`" in the naming. `NISimuCI` may be used in direct integration, projection, or stochastic collocation schemes. It is presented in Listing 3.5. The mentioned parameter is of type `array<real>`. It can be set by the second constructor — already at the initialisation of the component instance — or later on by the method `set_param`. The method `set_param` enables one to reuse the same instance of the component when a different deterministic model should be simulated. This can be utilised for a better performance. The method `solve` simulates the deterministic model and reports its success or failure by a returned four byte integer (error flag). The outcome is of type `array<real>` and can be obtained by method `get_state`.

```
#define CTL_ClassTmpl NISimuCI, ( real ), 1
#include CTL_ClassBegin

    #define CTL_Constructor1 ( const string ), 1
    #define CTL_Constructor2 ( const string,                \
            const array<real> ), 2

    #define CTL_Method1 void, set_param, (                  \
            const array<real> ), 1
    #define CTL_Method2 int4, solve, (), 0
    #define CTL_Method3 void, get_state, (                  \
            array<real> ) const, 1

#include CTL_ClassEnd
```

Listing 3.5: *The component interface* `NISimuCI` *in CTL syntax.*

The constructors declare a so far unregarded input argument of type `string`. Nevertheless, this string is the most essential support for a separation between the general conceptions of a deterministic simulator component and the individual conceptions of its component implementation interfacing a (third-party) simulation code. The string is actually a filename pointing at a so-called *control file*. A control file comprises declarations required for an individual component implementation, and consequently it is individual, too. It is usually read during the initialisation of the component instance. A control file may, for example, declare input or output files corresponding to an interfaced third-party simulation code, or it may declare how the parameter or the outcome are to be understood. As a consequence, a concrete control file along with the binding of the corresponding deterministic simulator component implies at runtime the application-specific semantics of the problem to be solved. A concrete realisation of a control file in the case of a deterministic simulator component for a preliminary aircraft design is presented in Section 3.4.4.3.

`FELinSimuCI` reflects a detailed understanding of the mathematical description of the deterministic model: the deterministic model is geometrically discretised by finite elements, it is linear and, thus, represented by a linear system. The CI may be used by SGM-based schemes. `FELinSimuCI` is presented in Listing 3.6. The non-inherited methods may be denoted as intrusive ones, as they ask for functionality which usually a finite element based third-party simulation code does not provide through its user interface also when the functionality is probably implemented somewhere inside the code. A setting of the parameter — either through the constructor

or the inherited method `set_param` — causes the deterministic simulator component to reconstruct the linear system: when the parameter for instance indicates the material of the deterministic model the linear operator — namely the stiffness matrix — and probably the right-hand side are reconstructed.

```
#define CTL_ClassTmpl FELinSimuCI, (real,integer), 2
#define CTL_Extends ( NIDSimuCI<real> ), 1
#include CTL_ClassBegin

    /* constructors and methods from "NIDSimuCI" */

    #define CTL_Method1 void, get_rhs, (                    \
            array<scalar1> ) const, 1
    #define CTL_Method2 void, get_stiff, (                  \
            array<integer>, array<integer>,                 \
            array<real> ) const, 3
    #define CTL_Method3 void, multiply, (                   \
            const array<real>, array<real> ) const, 2
    #define CTL_Method4 int4, solve_lin, (                  \
            const array<real>, array<real> ) const, 2

#include CTL_ClassEnd
```

Listing 3.6: *The component interface `FELinSimuCI` in CTL syntax.*

Iterative numerical schemes require a multiplication of the stiffness matrix with a vector to solve the system derived by the SGM. Such multiplications are repeated within the iterations. The method `multiply` provides the mentioned multiplication: the current stiffness matrix is multiplied by the first argument of type `array<real>`, the outcome is stored in the second argument of the same type. Method `get_stiff` provides the stiffness matrix itself in a sparse format: the first and second arguments of type `array<integer>` obtain the row and column indices of the stiffness matrix, while the third argument of type `array<real>` obtains the proper matrix entries. This method can be used for direct schemes to solve the system derived by the SGM. However, the obtained stiffness matrix can also be used to determine the matrix-vector product by itself outside of the simulator component. Obviously, there are two possibilities to perform the matrix-vector product. There are reasons to prefer one to the other, commented as follows. For instance, an uncertain material parameter causes the construction of potentially many stiffness matrices. When the number and dimension of the matrices are relatively small, so that all matrices can be located in the main memory simultaneously, method

`get_stiff` could be called to obtain these matrices initially and to store all of them temporarily. Then, the matrix-vector multiplications repeated within the iterations of the numerical scheme could be determined quickly without comparably expensive reconstructions (and remote communications). On the other hand a stiffness matrix could be of large dimension, so that only few of such matrices could be located in the main memory simultaneously. Then, the communication overhead is less when using `multiply` instead of `get_stiff` because the same matrices are repetitively invoked within the iterations.

A further method to provide the right-hand side of the linear system by an instance of type `array<real>` is `get_rhs`. The right-hand side of the deterministic model is used to construct the right-hand side of the probabilistic model.

Method `solve_lin` solves a linear system, at which the operator of the left-hand side is given by the current stiffness matrix. The right-hand side is passed by the first argument of the method; the outcome is stored in the second argument of the same type `array<real>`, and an error flag is returned. This method is especially introduced for the VLR-SR1U implementation. More precisely, it is introduced to solve the first linear system in Eq. (2.73) of Section 2.7.2.1 for the special case considered in Eq. (2.74), in which the stiffness matrix is linear in its material: the left-hand side can be set by `set_param`, the right-hand side would be the input argument of method `solve_lin`. Then, the communication overhead is kept low, as the linear operator is not transferred.

3.4.4.2 Interfaced Third-Party Simulation Codes for Deterministic Models

Concrete component implementations for the CIs of the deterministic simulator components applied or developed in this thesis are outlined in this section. Each of them interfaces a different third-party simulation code for deterministic models. The component implementation *coFeap* [96] implements `FELinSimuCI` (and therefore also the inherited CIs `NIDSimuCI` and `NISimuCI`) in its functional specifications and interfaces the finite element based program *FEAP* (*A Finite Element Analysis Program*) [199]. coFeap is used in the simulation of a groundwater flow problem in Section 4.1. A further component implementation for `FELinSimuCI` interfaces the finite element based program *ParaFEP* (*Parallel Finite Element Program*) [152]; an independent name for this component implementation is not assigned, it is an integral part of ParaFEP. It is used in the simulation of a laminated composite structure in Section 4.2.1.

The component implementation *coPrADO* implements `NISimuCI` in its functional specifications and interfaces the program *PrADO* (*Preliminary Aircraft Design and Optimisation program*) [82, 81] to simulate the design process of an aircraft. In contrast to the other component implementations coPrADO is the author's own contribution and therefore discussed in more detail in Section 3.4.4.3. It is applied in the numerical experiments of Section 4.3.

Application	groundwater flow	structural mechanics	aircraft design
Generic CI	`FELinSimuCI`	`FELinSimuCI`	`NISimuCI`
Implementation	coFeap	ParaFEP	coPrADO
Program	FEAP	ParaFEP	PrADO

Table 3.1: *Deterministic simulator components, their CIs, their component implementations, their interfaced third-party simulation codes (programs), and their fields of application in the numerical experiments of this thesis.*

A code may be compiled for different specialisations of the generic types to obtain different specialised distributed software components which can be bound at runtime. Table 3.1 summarises the mentioned component implementations and allocates them to the implemented CIs and the interfaced deterministic third-party simulation codes. Figure 3.6 illustrates the simulator components, their CIs, their component implementations, and their interfaced third-party simulation codes as well as their embedding in the software architecture; the figure also puts emphasis on the separation between the general conceptions through the CIs and the individual conceptions through the component implementations.

3.4.4.3 coPrADO: An Example of a Component Implementation

coPrADO is a component implementation for the deterministic simulator component specified by the component interface `NISimuCI`. It interfaces the program PrADO to simulate a preliminary aircraft design. This section exemplifies how the general conceptions of the CI are resolved by the individual conceptions of the component implementation in respect of a PrADO-based simulation. The next paragraph outlines parts of the program structure of PrADO relevant for the component implementation, the subsequent paragraph concentrates on coPrADO. (Numerical experiments involving coPrADO are presented in Section 4.3. Further information on aircraft design in general and the preliminary design phase in particular is presented in Section 4.3.1.)

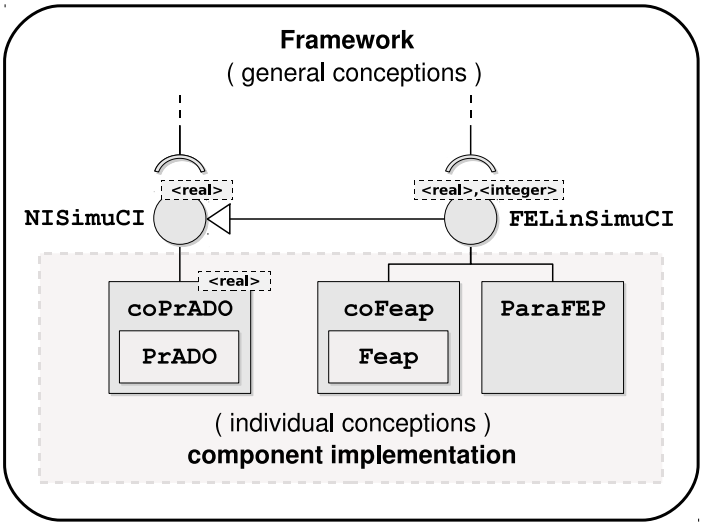


Figure 3.6: *Applied component implementations of deterministic simulator components: each implementation interfaces another third-party simulation code.*

PrADO: The Program Structure The program package PrADO [82, 81, 190, 211] offers a computer-aided simulation for the preliminary design of an aircraft. It includes many hundreds of modules, which exclusively communicate via a data management system called *DMS* and are executed by system calls. The modularised structure enables a problem-oriented interconnection of modules. In this context, modules can be replaced or new ones can be added. This emphasises the flexibility and expandability of PrADO. An aircraft is described by a parameterised model in PrADO; the corresponding parameters are henceforth called configuration parameters of the aircraft model. The set of the configuration parameters is thematically divided into several data banks of the DMS. The parameterised model in PrADO and its iterative computation is explained in Section 4.3.2.

A data bank of the DMS is based on an ASCII text file. A common entry of a data bank is a configuration parameter of the appropriate aircraft model. Listing 3.7 presents an extract of a data bank. A configuration parameter is formatted as follows. It consists of at least three lines. The first line introduces the configuration parameter by "<–" and its name, followed by its measurement unit and its meaning. The second

line contains some standard data mostly not relevant for the component implementation; the fourth integer assigns the size of the actual data vector corresponding to the configuration parameter. The actual data vector starts in the third line and may extend over successive lines.

```
1 <-WOE   kg   Operating kerb weight (w/ containers)
2     0    3    1    1
3   23927.7925713
4 <-VTKK1F1  m**3  vol. distr. (fuel tank 1, aerofoil 1)
5     0    3    1   10
6   1.70702240348    1.56984956932    1.36745998131
7   1.15732764562    0.94719531000    0.73706297459
8   0.52693064009    0.31679831532    0.12539989086
9   0.0
```

Listing 3.7: *An extract of a data bank of the DMS of PrADO corresponding to an appropriate aircraft model:* `WOE` *and* `VTKK1F1` *are configuration parameters;* `WOE` *is a scalar, while* `VTKK1F1` *is a vector (numbers are abridged).*

For further purposes within the scope of the component implementation, an entry of a data vector is referenced by row and column indices. A row marks a line, and its index is declared relatively to the introduction line of the corresponding configuration parameter. Both the row and column indices start at 1. For instance the entry of the configuration parameter `VTKK1F1` in the fourth row and the third column is `0.73706297459`.

coPrADO: The Component Implementation coPrADO is a `C++` component implementation consistent with the functional specifications of the component interface `NISimuCI`. It was developed by the author of this thesis. coPrADO can be instantiated to enable a remote access to a PrADO instance. It may be accessed in concurrency to other coPrADO instances (and consequently PrADO instances), which may also be located on other remote machines. As long as PrADO keeps its DMS — that means especially the format of the data banks — in later releases, coPrADO is invariant to changes within PrADO. That also means that properties of PrADO, like flexibility and expandability mentioned in the previous paragraph, are not affected by coPrADO. These features of coPrADO are possible because it works exclusively on the data banks and executes PrADO by a system call. The overhead induced by coPrADO is negligible, as system calls and of course the DMS are in any case integral parts of PrADO, which are usually used many times during an execution of PrADO. In addition, the actual runtime is typically dominated by the computations of PrADO.

The (generally held) parameter of the deterministic simulator component to specify a deterministic model is passed either through the second constructor or the method `set_param` of the component interface `NISimuCI` (see Section 3.4.4.1). It gets a concrete (individual) application-specific semantics through the component implementation coPrADO. Nevertheless, this application-specific semantics is only known by the component implementation and not by the software architecture. In its concrete application-specific semantics the parameter encapsulates individual configuration parameters (or parts of them) corresponding to an aircraft model in PrADO. (These configuration parameters are the uncertain ones inducing the probabilistic aircraft model.) A corresponding control file — already introduced in Section 3.4.4.1 — specifies how the parameter of the CI encodes configuration parameters of the appropriate aircraft model. Analogously, a control file specifies which individual configuration parameters of the aircraft model form the outcome provided by the method `get_state` of the CI. (A control file additionally enables one to specify a validation of the outcome, which is then reflected by the error flag returned through method `solve` of the CI; however, this is not further regarded in this section.) During the initialisation phase coPrADO reads the control file to understand the overall individual specifications.

A control file is an ASCII text file. It exhibits a special format, so that especially the parameter and the outcome of the deterministic simulator component are individually configurable concerning the aircraft model. Each file which satisfies that format is a valid control file. A valid control file (or more precisely an extract of it) is presented in Listing 3.8 (it is used in the numerical experiments of Section 4.3).

Some entries in the control file are not otherwise specified here. These are path declarations for the home directory and input and output files (the data banks) of PrADO, as well as the specification for the outcome validation. The parameter of the CI is represented by a vector with an index starting at 0. The entries of the parameter vector are specified in lines 9–13, each line means one entry (some lines are left out for reasons of clarity). Line 9, for instance, assigns the entry at the third row and the first column of configuration parameter `ERRWFL` to the entry at index 0 of the parameter vector. The size of the parameter vector is 14; each of the entries corresponds to an individual scalar configuration parameter of the aircraft model. Consequently, fourteen configuration parameters are considered to be uncertain. The flexibility of a control file is obvious: it enables one to set an individual sequence of configuration parameters of the aircraft model, each configuration parameter can be set partially or completely. Nevertheless, this also reveals a weakness: when a configuration parameter shall be set completely and its size is quite large many lines are required in the control file. This weakness could be solved by an automatic

```
1  #  ... path declarations in general      ...
2  #  ... path declarations for input files ...
3
4  #  ================================================
5  #  | PrADO input files              | CI          |
6  #  ================================================
7  #  | key      | row      | column   | idx(params) |
8  #  ================================================
9     ERRWFL     3          1          0
10    ERRWR      3          1          1
11    ERRWAT     3          1          2
12    #  ... more specifications follow ...
13    ERRTET     3          1          13
14
15
16 #  ... path declarations for output files ...
17 #  ... validation specifications          ...
18
19 #  ================================
20 #  | PrADO output files           |
21 #  ================================
22 #  | key      | row      | column |
23 #  ================================
24    IITVMAX    3          1
25    R2         3          1
26    POTW       3          1
27    #  ... more specifications follow ...
28    MPKTBLCLD  3          1
```

Listing 3.8: *A control file for coPrADO: a concrete application-specific semantics of the parameter declared in the component interface* `NISimuCI` *is defined in lines* 9–13 *for an appropriate aircraft model; lines* 24–28 *analogously do the same for the outcome. Some lines are left out for reasons of clarity.*

generation of control files. The specification of the outcome provided by method `get_state` of the CI is analogously stated in lines 24–28.

Now the general process of coPrADO is commented. The integral parts of coPrADO are demonstrated in Figure 3.7. The path of a valid control file is passed through a constructor of the component interface `NISimuCI`. During the initialisation phase a coPrADO instance reads the control file. When the second constructor or the method

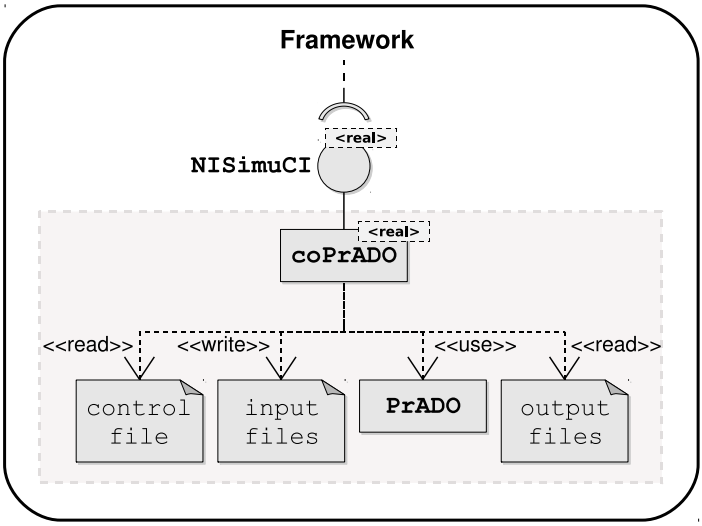


Figure 3.7: *Component implementation coPrADO embedded in a framework of the software architecture.*

`set_param` is called the provided sample of the parameter corresponding to the deterministic simulator component is written to the proper entries in the data banks of the DMS. In this manner the configuration parameters of the aircraft model are set. Method `solve` executes PrADO by a system call to simulate (solve) the configured deterministic model of an aircraft. The method `get_state` reads the configuration parameters — which form the outcome of the simulation — from the data banks of the DMS and stores them in its output argument.

3.4.4.4 Conclusion

The simulation of a probabilistic model implicates the construction and potentially the simulation of deterministic models. The simulation of deterministic models is not a recent discipline (compared to probabilistic models). As a consequence, a recommended corresponding simulation code is usually available for the application-specific field of interest and probably developed by a third party. However, the structure and handling of such codes are quite individual.

A deterministic simulator component is a distributed generic software component which reflects the general conceptions of a deterministic model and its handling through the CI. Its component implementation reflects the remaining individual conceptions, and interfaces a proper (third-party) simulation code. This component induces the proposed distributed generic component-based software architecture to be invariant to the application-specific semantics of the probabilistic model and structurally individual deterministic simulation codes until runtime. The individual conceptions are configured through a control file locally accessible for the deterministic simulator component. A pointer to this file is passed at the instantiation of the component. A concrete configuration of the control file and the binding of the corresponding specialised deterministic simulator component at runtime induce the concrete application-specific semantics and interface a concrete (third-party) simulation code. Furthermore, different CIs for deterministic simulator components are provided which exhibit an inheritance hierarchy. The inheritance hierarchy expresses an increasing demand (or increasing preconditions) on the deterministic model and its handling made by different kinds of numerical schemes. Direct integration or stochastic collocation schemes only ask for a configuration and simulation of a deterministic model. In this context, the deterministic model is — in a mathematical sense — understood as a black box. These demands are usually satisfied by each (third-party) simulation code. In contrast, an SGM-based numerical scheme postulates a detailed understanding about the mathematical description of the deterministic model and asks for more internal access. Section 3.4.4 discusses the component interfaces of deterministic simulator components and the technical separation of general and individual conceptions in more detail. (The inheritance relations are mainly addressed in Section 3.4.3.1.)

In this thesis, different third-party simulation codes are used for probabilistic models concerning different application-specific semantics, see Section 3.4.4.2. The implementation of a deterministic simulator component interfacing a third-party simulation code is exemplified for a preliminary aircraft design, see Section 3.4.4.3. In this context, the realisation of the individual conceptions is demonstrated, and a concrete realisation of the control file is presented.

3.4.5 Generic Implementation of the VLR-SR1U, VLR-OPT, and RBSSE Algorithms

Each of the VLR-SR1U, VLR-OPT, and RBSSE algorithms is implemented by a template class. The appropriate template parameters of such a template class reflect disjunct concerns among themselves. Furthermore, the concepts for the template

parameters are defined such that the template classes are invariant to different representations for matrices — corresponding to the system to be solved — and their specific handling as well as different discrete representations of an uncertain location-dependent material parameter. These conceptions — of disjunct concerns and particular invariance — enable one to configure the template classes at compile-time, so that the implementation is individually adapted to a specific runtime environment or specific properties of the system to be solved (which can be exploited for an efficient processing). A configuration happens through specialised classes satisfying the concepts for the template parameters. The template classes are part of the SGM-based framework presented in Section 3.4.3.3.

The algorithms are designed for a stationary linear elliptic boundary value problem with an uncertain location-dependent material parameter, which is structurally similar to the stationary groundwater flow problem with an uncertain hydraulic conductivity in Section 4.1, or the laminated composite structure with an uncertain material in Section 4.2. The mentioned stationary groundwater flow problem is described by Eqs. (2.34)–(2.36) in Section 2.2. Its discretisation — or the discretisation of structurally similar problems — by the SGM yields a linear system as in Eq. (2.61) or (2.64) of Section 2.6.2. The difference of these two linear systems is simply caused by different discrete representations of the hydraulic conductivity. The linear system in Eq. (2.64) is representatively considered in this section, unless otherwise stated. Its known integral parts are the operator and the right-hand side $\boldsymbol{F}$. The operator is separately described by integral parts following from the geometric (spatial) discretisation — namely the matrices $\{\boldsymbol{K}_i\}_{i\in\{0,\ldots,l\}}$ — and parts following from the stochastic discretisation — namely the matrices $\{\boldsymbol{\Delta}_i\}_{i\in\{0,\ldots,l\}}$.

The algorithms — in which way ever — operate with the matrices $\{\boldsymbol{K}_i\}_{i\in\{0,\ldots,l\}}$, $\{\boldsymbol{\Delta}_i\}_{i\in\{0,\ldots,l\}}$, and $\{\boldsymbol{F}\}$. The operations on one of these sets indicate one concern, so that in whole three disjunct concerns are considerable. The template parameters of the implemented template classes encapsulate these operations. The encapsulation of basic operations, on which more complex operations are depending, and the mentioned possibility of an individual configuration are demonstrated in Section 3.4.5.1. The relations between the template classes and the encapsulation of more complex operations are presented in Section 3.4.5.2. Section 3.4.5.3 summarises the overall content.

3.4.5.1 Encapsulation of the Basic Operations

Template class `VLR_SR1U` implements the VLR-SR1U algorithm in its basic and extended forms (the extended forms are induced by the internal use of the VLR-

OPT and RBSSE algorithms). It defines the template parameters `Gop`, `Sop`, and `F`. Each of these template parameters encapsulates the basic operations — mainly matrix-vector multiplications — with the matrices of its concern. In that regard, `Gop` encapsulates the basic operations with the matrices $\{\boldsymbol{K}_i\}_{i\in\{0,\ldots,l\}}$. Analogously, `Sop` focuses on the matrices $\{\boldsymbol{\Delta}_i\}_{i\in\{0,\ldots,l\}}$, and `F` focuses on matrix $\boldsymbol{F}$. The concept for each of these template parameters reflects the corresponding basic operations through method declarations. These method declarations do not reveal matrix representations. Quite the contrary, the matrices and of course their actual handling are hidden in the implementation of specialised classes satisfying the concept for `Gop`, `Sop`, and `F`.

The disjunct concerns and the independence from matrix representations and their specific handling result in a quite flexible implementation of `VLR_SR1U`, which is individually configurable at compile-time in each concern separately, so that an individual adaption to a specific runtime environment or specific properties of the system to be solved is possible. In other words, the implementation of a specialised class — satisfying the concept for a template parameter — is motivated by the wish or need to configure `VLR_SR1U` concerning the mentioned circumstances. This motivation is clarified in the next paragraphs by discussing some specialised classes satisfying the concept for `Gop` and `F`. The considered specialised classes for `Gop` are motivated by an adaption to the runtime environment, while the specialised classes for `F` are motivated by different specific properties of the system to be solved. Specialised classes for `Sop` are comparable to the ones for `Gop` and shortly addressed in the paragraph for `Gop`.

Specialisations for the Template Parameter `Gop` of Template Class `VLR_SR1U` Different specialisations for the template parameter `Gop` of template class `VLR_SR1U` are provided through template classes `SGM_gop` and `SGM_gop_fr`. The motivations for these specialised classes relate to a configuration of `VLR_SR1U` in terms of different runtime environments. In this regard, `SGM_gop` assumes a set of large-sized matrices $\{\boldsymbol{K}_i\}_{i\in\{0,\ldots,l\}}$ not anymore storable in the local hard disk drive, while `SGM_gop_fr` assumes this set to be at least storable in the local hard disk drive but not in the main memory. These individual characteristics are reflected by the constructor declarations and, needless to say, the implementations of the specialised classes, whereas the method declarations match the concept for template parameter `Gop`. First of all, the template class `SGM_gop` is discussed, then — at the end of this paragraph — the motivation for each specialised class is considered in more detail.

Listing 3.9 shows the declaration of template class `SGM_gop` in extracts. While template parameter `Gop` of `VLR_SR1U` administrates the handling of each of the

potentially numerous matrices $\{\boldsymbol{K}_i\}_{i\in\{0,\ldots,l\}}$, template parameter `Op` of `SGM_gop` only handles exactly one of these matrices. Nevertheless, the concept for `Op` also declares a method `set_param` to reconfigure the matrix. The component interface `FELinSimuCI` — discussed in Section 3.4.4 — includes the method declarations prescribed by the concept for `Op`. As a consequence, a corresponding deterministic simulator component is a specialisation for template parameter `Op`. In this manner the possibility of a distribution of processes enters the proposed implementation of the VLR-SR1U algorithm. The second template parameter `VVr` of `SGM_gop` is meant to be a vector of vectors of real values. `Vr` is its value type. When `VVr` is, for instance, specialised by `std::vector<std::vector<double> >` `Vr` is `std::vector<double>`.

```
1  template<class Op, class VVr> class SGM_gop {
2
3      typedef typename VVr::value_type     Vr;
4      typedef typename Vr::value_type      real;
5
6  public:
7      SGM_gop( const VVr* mparams, Op* op );
8
9      unsigned long get_n() const;
10     void multiply ( unsigned long i,  const Vr& x,
11             Vr& y ) const;
12     void solve_lin( const Vr& mparam, const Vr& y,
13             Vr& x ) const;
14
15     /*...Some other methods and attributes...*/
16 };
```

Listing 3.9: *An extract of the declaration of template class `SGM_gop`: the template class `SGM_gop` is a specialisation for the template parameter `Gop` of template class `VLR_SR1U`.*

The constructor of `SGM_gop` has two arguments. The second argument points at an object `op` of type `Op`. The first argument `mparams` comprises the deterministic vectorial parameters corresponding to a discrete representation of an uncertain location-dependent material parameter. The discrete representation may be a truncated KL or PC representation. Then, the vectorial parameters are either the KL eigenvectors or the PC coefficient vectors. The vectorial parameters are indexed from 0 to $n-1$ and accessed through that index in the method declarations prescribed by the concept for `Gop`. As a consequence, (the concept for) the template parameter `Gop` is invariant to

different discrete representations of the material parameter, and so is VLR_SR1U.

Method get_n of SGM_gop returns the number of deterministic vectorial parameters corresponding to the discrete representation of the material parameter. The method multiply enables the multiplication $\boldsymbol{y} := \boldsymbol{K}_i \cdot \boldsymbol{x}$, at which the matrix $\boldsymbol{K}_i$ is specified by the vectorial parameter indexed through method argument i. The method solve_lin solves a linear system $\boldsymbol{K}_{mparam} \cdot \boldsymbol{x} = \boldsymbol{y}$, at which the operator is specified by a discrete material sample passed through the argument mparam (the sense of this method is explained in Section 3.4.4.1 in the context of component interface FELinSimuCI.)

As already mentioned, SGM_gop follows a concrete motivation. It assumes that the size of a stiffness matrix $\boldsymbol{K}_i$ is so large that the capacity of the local hard disk drive is not big enough to store all of them. Whenever a stiffness matrix is required — for instance when invoking method multiply — it is reconstructed by the — maybe remotely located — instance op.

In contrast, template class SGM_gop_fr assumes that the size of a stiffness matrix $\boldsymbol{K}_i$ is only too large to keep all of them in the local main memory. Stiffness matrices $\{\boldsymbol{K}_i\}_{i\in\{0,\ldots,l\}}$ are precomputed and separately stored in files. The constructor of SGM_gop_fr obtains the corresponding filenames. Whenever a stiffness matrix is required it is read from a file. A further specialisation for template parameter Gop is obvious: when the size of a stiffness matrix $\boldsymbol{K}_i$ is relatively small all of them can be kept in the main memory. Hence, various motivations are conceivable to implement specialisations for Gop.

Specialisations for template parameter Sop are likewise implemented. The software component SBasisCI — introduced in the SGM-based framework (see Section 3.4.3.3) — takes on the role for the specialisations of Sop, while FELinSimuCI takes on the role for the specialisations of Gop.

Specialisations for the Template Parameter F of Template Class VLR_SR1U Different specialisations for the template parameter F of template class VLR_SR1U are provided by template classes SGM_f and SGM_f_debc. These specialised classes are only shortly presented, as they are structurally comparable to the ones in the previous paragraph. However, the motivations for them relate to a configuration of VLR_SR1U in terms of different specific properties for the right-hand side of the system to be solved. In this regard, SGM_f concerns homogeneous essential boundary conditions of value zero, a deterministic source/sink term, and

deterministic natural boundary conditions. This configuration of the right-hand side implicates the first column of $\boldsymbol{F}$ to be the only non-zero column. This enables an efficient implementation of the basic operations with $\boldsymbol{F}$. In contrast, `SGM_f_debc` concerns deterministic inhomogeneous essential boundary conditions and assumes the source/sink term and the natural boundary conditions to be zero.

3.4.5.2 Template Classes

The VLR-SR1U, VLR-OPT, and RBSSE algorithms are implemented by template classes `VLR_SR1U`, `VLR_OPT`, and `RBSSE`. The template parameters of each of these template classes encapsulate operations with the integral parts of the system to be solved, so that the template classes are invariant to matrix-representations and their specific handling. The template parameters of `VLR_SR1U` encapsulate the basic operations on which more complex operations are depending — this and the associated strengths are explicitly discussed in Section 3.4.5.1. The template parameters of the other template classes encapsulate the already signified more complex operations. Further template classes are introduced to handle similarities between the algorithms. Figure 3.8 points out the instantiations between the involved template classes and emphasises proper specialisations for the appropriate template parameters. The VLR-SR1U and VLR-OPT algorithms are exclusively considered, the template implementation of the RBSSE algorithm follows an analogous structure.

The VLR-OPT algorithm applies the conception of the VLR-SR1U algorithm onto projected systems instead of the original system. Nevertheless or precisely because of that both algorithms share the mechanism of a rank-one update. Template class `VLR_SR1U_r1u` undertakes the implementation of a rank-one update and is instantiated by `VLR_SR1U` and `VLR_OPT`. The different systems — if original or projected — are encapsulated through the template parameter of `VLR_SR1U_r1u`.

In other words, `VLR_SR1U_r1u` is a key template class for `VLR_SR1U` and `VLR_OPT`. The concept for the template parameter `Op` of `VLR_SR1U_r1u` prescribes the methods `solve_for_lv` and `solve_for_rv`. Method `solve_for_lv` approximates the left vector corresponding to the approximated right vector of the current rank-one update; analogously, method `solve_for_rv` approximates the right vector corresponding to the approximated left vector of the current rank-one update. `VLR_SR1U_r1u` alternatingly invokes both methods to determine a rank-one update.

`VLR_SR1U` instantiates `VLR_SR1U_r1u` with the proper specialisation

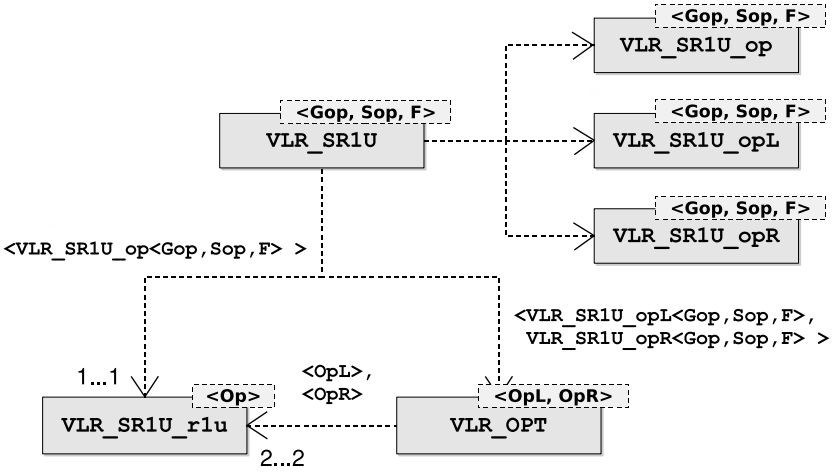


Figure 3.8: *Template classes implementing the VLR-SR1U and VLR-OPT algorithms: a template class is connected to another template class when it instantiates the other one; a template specialisation and multiple instantiations may be declared next to a connector.*

`VLR_SR1U_op` for the template parameter `Op` to realise the basic VLR-SR1U algorithm. The method `solve_for_lv` of `VLR_SR1U_op` solves only the first equation in Eq. (2.73), analogously method `solve_for_rv` of `VLR_SR1U_op` solves only the second equation in Eq. (2.73) (see Section 2.7.2.1). The template parameters of `VLR_SR1U_op` themselves match with the ones of `VLR_SR1U` already discussed in Section 3.4.5.1.

`VLR_OPT` instantiates `VLR_SR1U_r1u` twice times: `Op` of `VLR_SR1U_r1u` is specialised once by template parameter `OpL` and once by template parameter `OpR` of `VLR_OPT`. The corresponding concept prescribes for `OpL` the methods `solve_for_lv` and `solve_for_rv` — but in contrast to `VLR_SR1U` — with the emphasis on the original system projected onto the right rank-one update vectors. In other words, the methods `solve_for_lv` and `solve_for_rv` declared by template parameter `OpL` of `VLR_OPT` solve the first and second equations in Eq. (2.78) (see Section 2.7.2.2). Analogously, the methods `solve_for_lv` and `solve_for_rv` of template parameter `OpR` focus the original system projected onto the left rank-one update vectors; consequently, the methods solve the first and second equations in Eq. (2.80).

Template class VLR_SR1U implements the basic VLR-SR1U algorithm, but also enables one to optionally apply VLR-OPT and RBSSE; for this purpose, VLR_SR1U instantiates VLR_OPT and RBSSE. (VLR_OPT and RBSSE are independent of VLR_SR1U and may be used beyond the implementation of the VLR-SR1U algorithm.) VLR_SR1U instantiates VLR_OPT to enable an optimisation of a current low-rank approximation so that the expectation of the total potential energy can be really minimised. In that regard, proper specialisations for the template parameters OpL and OpR are the template classes VLR_SR1U_opL and VLR_SR1U_opR. The template parameters of VLR_SR1U_opL and VLR_SR1U_opR themself match with the ones of VLR_SR1U already discussed in Section 3.4.5.1.

3.4.5.3 Conclusion

The implementations of the VLR-SR1U, VLR-OPT, and RBSSE algorithms include a number of template classes. (The implementations are used in the developed SGM-based framework introduced in Section 3.4.3.3.) The known integral parts of the mathematical system to be solved are divided into three disjunct concerns, namely for the right-hand side, the parts of the operator associated with the discretisation of the geometrical domain, and the parts of the operator associated with the discretisation of the stochastic domain. Each integral part identifies a matrix. The mentioned algorithms operate with the integral parts. Template parameters are introduced to encapsulate these operations and hide matrix representations and their specific handling.

Template class VLR−SR1U implements the VLR-SR1U algorithm. It declares template parameters separately for the disjunct concerns. A template parameter encapsulates the basic operations with the integral parts of its concern, see Section 3.4.5.1. As a consequence, the implementation is invariant to the representation and specific handling of the matrices. The concept of the disjunct concerns and the invariance enable one to configure the template class quite individually at compile-time in each disjunct concern through the template parameters. A special configuration may address a particular runtime environment, or it may address particular properties of known integral parts of the mathematical system which can be exploited for an efficient processing. Different specialised classes for the template parameters are presented to clarify the motivation for an individual configuration.

Further template classes share the template parameters of VLR−SR1U and implement more complex operations with the integral parts. These classes are in turn specialisations of the template parameters declared in the template classes implementing

the algorithms VLR-OPT and RBSSE, see Section 3.4.5.2. By so doing a strongly individual configuration is maintained also for these template classes.

3.5 Conclusion

Chapter 3 discusses the major characteristics of the distributed generic component-based software architecture developed in this thesis to simulate probabilistic models [109, 108, 110]. A key feature is the application of distributed generic software components. These components support a generic programming on the implementation level, and their compiled representatives are interchangeable at runtime. In this manner they carry over the generic support from the implementation level to the runtime level. A distributed generic software component is specialised by concrete data types for the generic types before compilation. The compilation results in a specialised distributed software component bound at runtime. Distributed generic software components are discussed in Section 3.3.4.

The simulation of probabilistic models identifies many different concerns. These are, for example, the modelling of uncertain parameters or the simulation or construction of deterministic models. Both — an uncertain parameter and a deterministic model — may introduce another concern, namely a discretisation of a potentially existing geometrical domain. Distributed generic software components are proposed to address these concerns. Familiar relations between component interfaces are expressed by inheritances. The developed distributed generic software components are used to compose frameworks. Each framework implements a special numerical scheme or a special class of numerical schemes. Frameworks for a direct integration, a stochastic collocation, or an SGM-based simulation are described here. The mentioned concerns, the associated distributed generic software components and composed frameworks are discussed in Section 3.4.3.

The simulation or construction of deterministic models is an essential integral part to simulate a probabilistic model. Recommended simulation codes — probably from a third-party — are usually available to process a deterministic model, but the external (and internal) handling of such codes may be quite individual. Nevertheless, the proposed software architecture is invariant to such individual simulation codes and even invariant to the application-specific semantics of the probabilistic model at least until runtime. This is achieved by a deterministic simulator component. It reflects the general conceptions for the handling of deterministic models through the component interface. Its component implementation handles the individual conceptions. The

application-specific semantics is introduced at runtime by the binding and configuration of a compiled specialised deterministic simulator component. In other words, an individual third-party code and, as a consequence, the application-specific semantics can be interchanged at runtime. Different third-party simulation codes are applied in this manner for simulating probabilistic models of different application-specific semantics. Deterministic simulator components in general and in particular are discussed in Section 3.4.4, the mentioned inheritance relations are separately addressed in Section 3.4.3.1.

The VLR-SR1U algorithm approximates a low-rank solution of a mathematical system. This system is also the basis for the VLR-OPT and RBSSE algorithms. Its known integral parts can be divided into three disjunct concerns, namely the right-hand side, the geometric part of the operator, and the stochastic part of the operator. The integral parts are matrices with which the algorithms operate. The algorithms are implemented by template classes and used in the provided SGM-based framework. The template parameters encapsulate operations with the integral parts and reflect the disjunct concerns. In doing so, the implementations are configurable in each disjunct concern separately. A configuration may simply enable the use of another internal matrix representation, or it may handle matrices remotely instead of locally. A configuration for the right-hand side may exploit special properties caused by the actual setting of the boundary conditions or the source/sink term. The implementations of the VLR-SR1U, VLR-OPT, and RBSSE algorithms are discussed in Section 3.4.5.

Chapter 4

Numerical Experiments

The numerical experiments in this chapter comprise problems regarding different application-specific areas. A groundwater flow problem with an uncertain hydraulic conductivity is chosen to demonstrate the promising convergence behaviour of the VLR-SR1U scheme in its different configurations and in comparison with direct integration schemes like (Q)MC methods and the Smolyak algorithm. For the other areas more emphasis is put on the modelling of corresponding uncertain parameters, which drive the uncertainty inside the corresponding problems to be solved. A model for a laminated composite material in a three-dimensional geometrical domain is discussed, which gets its input statistics from photographic images of a test specimen. A corresponding laminated composite structure is simulated, and resulting statistics demonstrate the need for the consideration of uncertainty. Finally, uncertain parameters are introduced to a design of a future civil aircraft to examine its robustness. Statistics are computed and an efficient surrogate model is constructed for the probabilistic model of the aircraft. The distributed generic component-based software architecture in Section 3 is used for the numerical experiments in this chapter; random numbers for an MC sampling are obtained by the generator RANDLIB [32].

Section 4.1 addresses the simulation of the uncertain groundwater flow problem. The laminated composite material and a corresponding structure are focused on in Section 4.2, and the uncertain aircraft design is discussed in Section 4.3. A conclusion is presented in Section 4.4.

4.1 A Stationary Groundwater Flow Problem

A stationary groundwater flow problem with uncertain hydraulic conductivity is simulated in this section through different numerical schemes which are compared with

respect to their convergence. The schemes are direct integration schemes like a basic MC method, a QMC method, and the Smolyak algorithm, but also the VLR-SR1U scheme. Here, a special emphasis is put to the VLR-SR1U scheme and its different configurations. It is demonstrated that this scheme belongs to the efficient ones.

The setting of the problem and general information on the simulation and the output handling are commented on in Section 4.1.1. Direct integration schemes are applied in Section 4.1.2. The basic SGM is used in Section 4.1.3 to obtain references with which the convergence of the VLR-SR1U scheme can be explored up to machine precision. The convergence of the VLR-SR1U scheme in its different configurations is in detail discussed in Section 4.1.4. A conclusion is presented in Section 4.1.5.

4.1.1 The Problem Setting and the Simulation in General

The stationary groundwater flow problem is presented in Eq. (2.34) ff. with an uncertain hydraulic conductivity κ. Here it is considered on a rectangular domain, see Figure 4.1. Deterministic inhomogeneous essential boundary conditions are applied on the left and right boundaries, natural conditions equal zero anywhere else. The source/sink term is assumed to be zero.

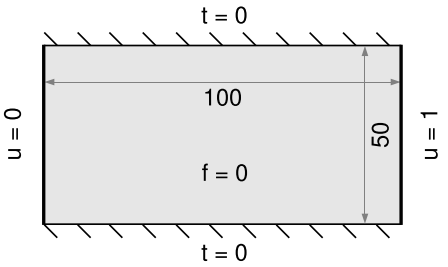


Figure 4.1: *The rectangular domain of the considered groundwater flow problem with deterministic inhomogeneous essential boundary conditions on the left and right boundary, natural boundary conditions equal zero anywhere else, and a zero source/sink term.*

The setting for the uncertain parameter κ is commented on in the next paragraph. Some general remarks about the simulation follow in the second paragraph. Outputs and corresponding references are discussed in the third and fourth paragraph.

Input Setting The uncertain parameter κ is lognormally distributed and represented in different ways, see Table 4.1. The first representation is $g := \exp(\gamma_m)$, at which γ_m is a truncated KLE of a Gaussian field γ. A truncated PCE κ_c of κ is determined from g (see Section 2.1.1.5), and a truncated KLE κ_l is then constructed from κ_c. As a consequence of the truncations, κ_c and κ_l are only approximations of g.

Representation g is applied in the direct integration schemes. Either κ_c or κ_l is used within the basic SGM, at which the resulting linear system is directly solved. Finally, κ_l is chosen for the VLR-SR1U scheme.

symbol	explanation	application
g	exponentiation of a Gaussian field represented by a KLE with m modes	(Q)MC, Smolyak
κ_c	PCE of g truncated under condition c	SGM
κ_l	KLE of κ_c with l modes	SGM, VLR-SR1U

Table 4.1: *Representations of the uncertain parameter κ and their applications.*

The input for γ is the expected value $\bar{\gamma}$ and the covariance function

$$cov_\gamma(\boldsymbol{x}, \boldsymbol{y}) := \sigma^2 \, e^{-\frac{(\|\boldsymbol{x}-\boldsymbol{y}\|_2)^2}{L^2}}$$

with variance $\sigma^2 = 1.0$ and correlation length $L = 20.0$. γ_m contains five terms (expectation excluded), that means $m = 5$. (m identifies the stochastic dimension of the problem.) Unless otherwise stated, κ_l contains as many terms, so that approximately 99.99% of the energy of κ_c is conserved: the energy conservation is estimated through the relation between the sum of the corresponding l eigenvalues and the trace tr ($\boldsymbol{MC}$), at which $\boldsymbol{M}$ is the mass matrix and $\boldsymbol{C}$ the covariance matrix, see Section 2.1.1.2 for more details.

The number of geometrical DoFs is $N_X = 21 \times 11 = 231$ (equal to the number of FE nodes). The small stochastic dimension and the coarse discretisation of the geometrical domain allow to directly solve the linear system derived by the SGM. In this manner references for solutions are obtained which enable one to examine the convergence behaviour of the VLR-SR1U scheme up to machine precision.

Solution The CTL component coFeap [96] is used as the deterministic simulator component for all experiments. Up to 144 distributed instances of that component are involved.

Output Setting Stochastic moments — like the first four absolute moments and the variance — are computed in the following numerical experiments. A moment is defined on the entire geometrical domain through N_X basis functions and the corresponding coefficients. A reduction function operates on a moment to obtain a scalar value for convergence considerations. Such a function may integrate the moment over the entire geometrical domain or a partial area, it may determine the arithmetic mean of corresponding coefficients, or may extract a specific coefficient. These different kinds of reductions are described by the set of functions $\{\mathcal{T}_i\}_{i\in\{1,\dots,5\}}$ with $\mathcal{T}_i : \mathbb{R}^{N_X} \to \mathbb{R}$, see Figure 4.2 for a graphic understanding. Function $\mathcal{T}_1$ or $\mathcal{T}_2$ calculates the arithmetic mean over the entire domain (marked in green) or at the vertical middle of the domain (marked in yellow). $\mathcal{T}_3$ or $\mathcal{T}_4$ integrates over the entire domain or over the mentioned vertical middle of the domain. $\mathcal{T}_5$ simply extracts the geometrical DoF (marked in light blue), which is most nearly located to $\frac{5}{8}$ of the horizontal domain size and $\frac{3}{4}$ of the vertical domain size. Table 4.2 summarises the functions and offers compact descriptions.

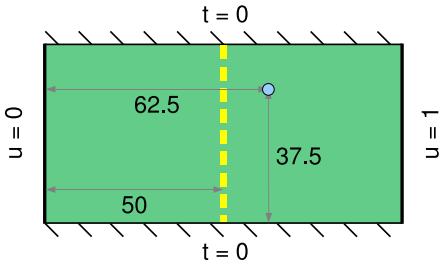


Figure 4.2: *Considered areas and locations of the rectangular domain of the groundwater flow problem: the entire domain (marked in green), the vertical middle (marked in yellow) and the location at $\frac{5}{8}$ of the horizontal domain size and at $\frac{3}{4}$ of the vertical domain size (marked in light blue).*

function	operation	area/location
$\mathcal{T}_1$	average	entire domain
$\mathcal{T}_2$	average	vertical middle
$\mathcal{T}_3$	integration	entire domain
$\mathcal{T}_4$	integration	vertical middle
$\mathcal{T}_5$	extraction	$\frac{5}{8}$ and $\frac{3}{4}$ of horizontal and vertical domain sizes

Table 4.2: *Functions to reduce a stochastic moment — defined on the geometrical domain — to a scalar value.*

References The convergence of the considered numerical schemes is displayed through the relative errors of the scalar outputs described in the previous paragraph. The references for these scalars are obtained in different ways, depending on the context. The overall references — valid for all applied numerical schemes — are obtained through a basic MC method and $328,195,000$ sample-points. These references are referred to as the MC references in the following sections. An empirical analysis by means of the central limit theorem (CLT) [4] estimates the error of a considered stochastic moment for each of the N_X spatial coefficients separately which leads to N_X separated errors. The probability that the computed errors are below a given threshold is larger than 0.9999. The maximum error of the separated errors is used as an error estimate for the scalar references corresponding to the stochastic moment. The CLT reveals that the error of a reference corresponding to one of the absolute moments is less than $5 \cdot 10^{-5}$; the error of a reference corresponding to the variance is less than $3 \cdot 10^{-4}$. As a consequence a consideration of relative errors is only meaningful up to the appropriate error bound. The function $\frac{1}{\sqrt{n}}$ or $\frac{1}{n}$ defined on the number n of sample-points may be displayed in convergence plots as an aid to orientation. $\frac{1}{\sqrt{n}}$ reflects the Monte Carlo convergence rate while the display of $\frac{1}{n}$ is motivated by the Quasi-Monte Carlo convergence rate, see Section 2.3.1.

Another two sets of references are determined through the basic SGM, once with parameter representation κ_c, and once with κ_l. In both cases the linear system — resulting from the SGM with a basic finite set of stochastic basis polynomials — is directly solved. As the system matrix grows immensely when increasing the dimension or the maximum order of the stochastic polynomials, direct solutions are only available for coarsely discretised problems. In this thesis the direct solving processes caused a main memory utilisation of up to 32 Gigabytes. The first three absolute moments and the variance are analytically computed from the direct solutions. The mentioned reduction functions are applied onto these moments to get scalar references. As a consequence of the direct solutions and the analytical computation of stochastic moments, these references enable one to consider the convergence behaviour of iterative SGM-based schemes — like the VLR-SR1U scheme — up to machine precision.

4.1.2 Direct Integration Schemes

The applied direct integration schemes are a basic MC method, the QMC method based on the Halton point sequence, and the Smolyak algorithm with different underlying one-dimensional quadrature rules. The Smolyak algorithm with the one-dimensional Kronrod-Patterson quadrature rule reveals the best convergence: while

for instance only a few hundred sample-points are enough to reach the reference solution, the QMC method requires ten thousands of sample-points, and the MC method even more.

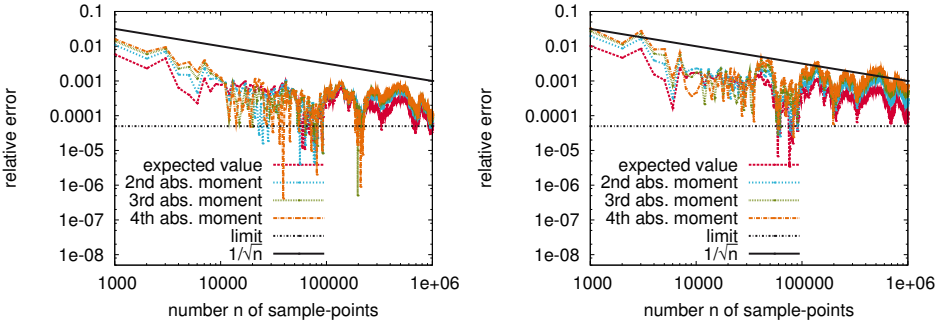


Figure 4.3: *Convergence of a basic MC method with respect to the first four absolute moments: the convergence is shown for reduction functions $\mathcal{T}_1$ in Figure (a) and $\mathcal{T}_2$ in Figure (b) (see Table 4.2). The MC references are used to compute the relative errors.*

The MC references — described in Section 4.1.1 — are used to compute relative errors in this section. The corresponding error bounds indicate the accuracy limits up to which a consideration of relative errors is meaningful. Below these limits no reasonable statement can be made.

The convergence of the MC method is presented in Figure 4.3. The QMC method based on the Halton point sequence is considered in Figure 4.4. The QMC method seems to be more robust with slightly better convergence than the MC variant.

The Smolyak algorithm is considered with respect to several underlying one-dimensional quadrature rules. The best convergence is obtained by applying the one-dimensional Kronrod-Patterson rule: this Smolyak configuration is abbreviated by "S-KP". It provides excellent convergence behaviour for all reduction functions and all four absolute moments (which are considered here). This is demonstrated in Figures 4.5(a) and (b). Mostly, the relative error of the scalar values extracted from the expected value is directly below the error bound — even with only one sample-point. S-KP requires a maximum of $1,471$ sample-points to reach the accuracies of all references. The previously discussed MC and QMC schemes are not competitive

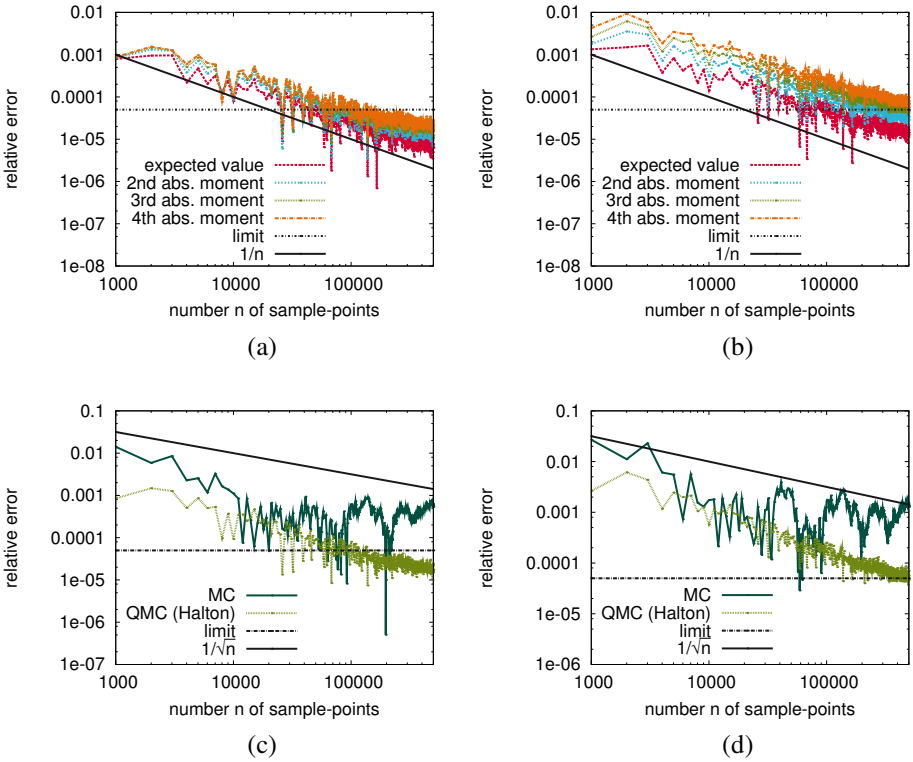


Figure 4.4: *Convergence of the QMC method based on the Halton point sequence with respect to the first four absolute moments: the convergence is shown for reduction functions $\mathcal{T}_1$ in Figure (a) and $\mathcal{T}_2$ in Figure (b) (see Table 4.2). The convergence of the QMC method in comparison to the one of the basic MC method is demonstrated in Figure (c) for $\mathcal{T}_1$ and in Figure (d) for $\mathcal{T}_2$ by considering in both cases the third absolute moment. The MC references are used to compute the relative errors.*

in these numerical experiments: the QMC method needs ten thousands to hundred thousands of sample-points, and the MC method needs even more sample-points. The second best results are provided by the one-dimensional Gauss-Hermite quadrature rule. It shows slightly inferior convergence and requires $7,778$ sample-points at the most to reach the accuracies of all references. A comparison with the S-KP is demonstrated in Figures 4.5(c) and (d).

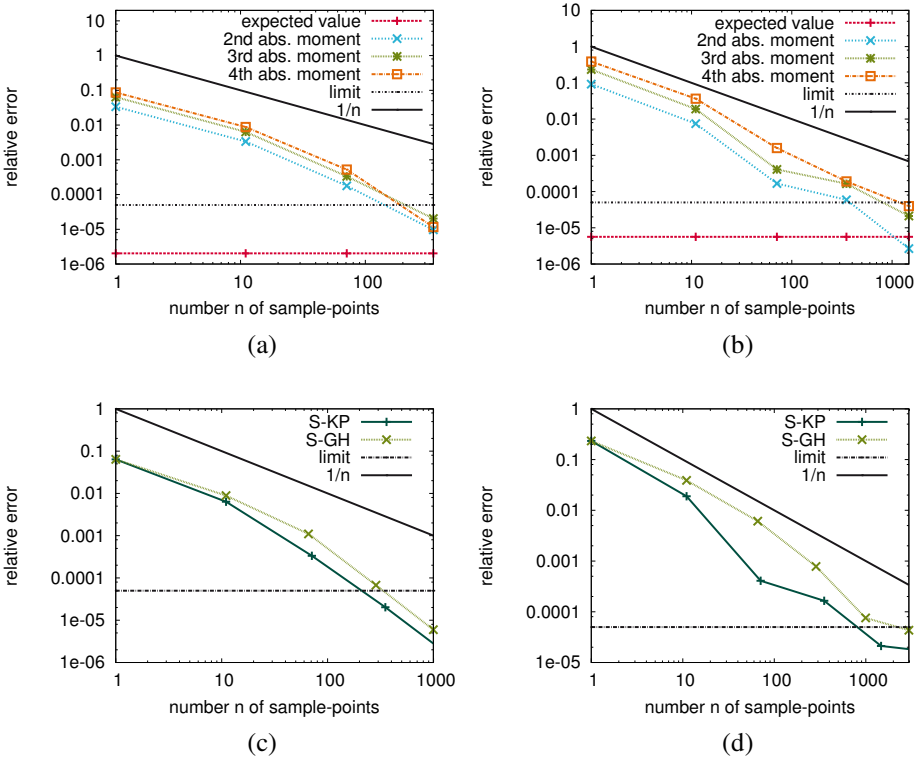


Figure 4.5: *Convergence of the Smolyak algorithm: the convergence with respect to the first four absolute moments is shown for reduction functions $\mathcal{T}_1$ in Figure (a) and $\mathcal{T}_2$ in Figure (b) (see Table 4.2) by using the Kronrod-Patterson rule (S-KP). The convergence of the Smolyak algorithm (S-GH) using the Gauss-Hermite quadrature rule in comparison to the one of S-KP is demonstrated in Figure (c) for $\mathcal{T}_1$ and in Figure (d) for $\mathcal{T}_2$ by considering in both cases the third absolute moment. The MC references are used to compute the relative errors.*

Convergence plots for further Smolyak configurations are neglected here, but some annotations are made. The delayed Kronrod-Patterson quadrature rule delivers acceptable convergence. It requires a maximum of $38,303$ sample-points to reach the accuracies of all references. A Clenshaw-Curtis configuration is only acceptable for some scalars, which are extracted from the first two absolute moments. For other

moments it requires hundreds of thousands of sample-points. The Smolyak algorithm based on Gauss-Legendre quadrature displays the slowest convergence and is not competitive at all.

The fast convergence of some Smolyak configurations may be traced back to the relatively small stochastic dimension of five. (Experiments which were carried out in higher dimensions (but are not reflected here) showed slower convergence of the Smolyak algorithm. Additionally, the Smolyak algorithm tended to non-robust behaviour, which may arise from negative weights occurring inside the algorithm.)

4.1.3 The Basic SGM

The SGM with a basic finite set of stochastic basis polynomials is applied to discretise the groundwater flow problem. The resulting linear system is solved by a direct scheme. Two different representations, namely a truncated PCE and a truncated KLE, are used for the description of the uncertain parameter κ. The corresponding convergence of the basic SGM is considered with regard to an increase of the maximum order for the polynomials which span the stochastic solution space. A rating of the stochastic polynomials demonstrates that a few of them describe most of the solution.

This section actually comprises a preparation for the VLR-SR1U scheme in Section 4.1.4. On the one hand the solutions obtained in this section are used as references in Section 4.1.4 to consider the convergence behaviour of the VLR-SR1U scheme up to machine precision. On the other hand the rating of the stochastic polynomials reveals the advantage of an adaptive solution space construction by the VLR-SR1U-ADAPT scheme.

The stochastic solution space is spanned by the polynomials which are indicated through the set $\mathcal{I}^{mod}$ of multi-indices, see Eq. (2.22) in Section 2.1.1.3. The consideration of the convergence is divided in two paragraphs. The PC representation κ_c (see Table 4.1) for the uncertain parameter is used in the first paragraph. In contrast, the KL representation κ_l is applied in the second paragraph.

PC Representation κ_c In this paragraph the uncertain parameter is discretised by the PC representation κ_c. The convergence of the basic SGM is presented in Figure 4.6. As already mentioned in the previous section, the error bounds inside the

plots identify the limits up to which a consideration of the appropriate relative error is meaningful.

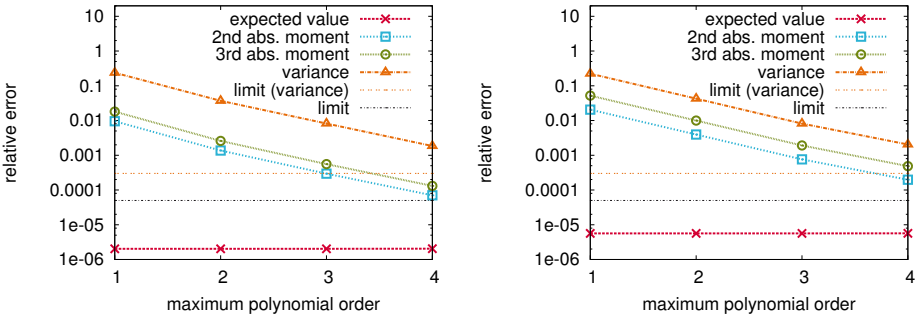


Figure 4.6: *Convergence of the basic SGM for the groundwater flow problem with parameter representation κ_c when the maximum order for the polynomials which span the stochastic solution space is increased: reduction functions $\mathcal{T}_1$ and $\mathcal{T}_2$ are applied onto some stochastic moments in Figures (a) and (b) (see Table 4.2). The MC references are used to compute the relative errors.*

The significance of each stochastic polynomial in describing the solution in a given solution space is measured here by the fractional variance which is contributed by each stochastic polynomial. The fractional variance $\hat{\sigma}_i^2$ contributed by the i'th stochastic polynomial is defined as

$$\hat{\sigma}_i^2 := \frac{\|\boldsymbol{\sigma}_i^2\|_2}{\sum_{j=1}^{N_S} \|\boldsymbol{\sigma}_j^2\|_2}. \tag{4.1}$$

$\boldsymbol{\sigma}_i^2$ is the contribution to the spatial point variance; the latter is composed by the variances at the discrete FE nodes. It is assumed that the N_S stochastic polynomials are sorted with respect to the fractional variances that means $\hat{\sigma}_i^2 \geq \hat{\sigma}_j^2$ for $i < j$. Then the cumulative fractional variance function can be formulated by

$$q(i) := \sum_{j=1}^{i} \hat{\sigma}_j^2. \tag{4.2}$$

Figure 4.7(a) shows the decline of the sorted fractional variances. The cumulative fractional variances are presented in Figure 4.7(b). It is obvious that few stochastic

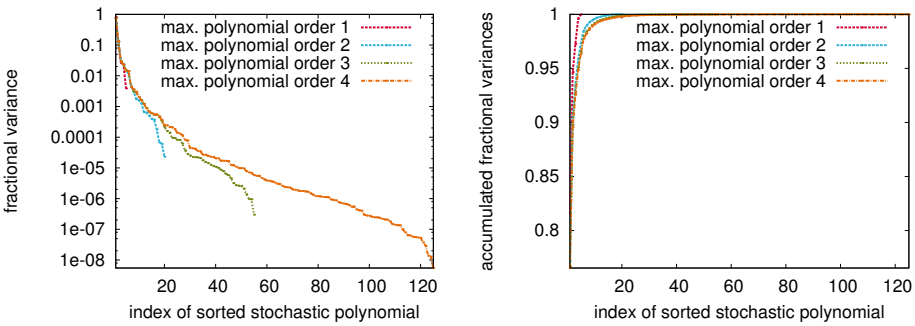


Figure 4.7: *The decline of the sorted fractional variances in Figure (a) and the increase of the cumulative fractional variances in Figure (b): the fractional variances are computed for PC representations of the solution up to fourth polynomial order.*

polynomials describe the most of the variance which the solution space is capable to resolve.

KL Representation κ_l In this paragraph the uncertain parameter is discretised by the KL representation κ_l. A truncation is done for different accuracies. The accuracy of a truncated KL representation is measured through the percentage of the conserved energy. On the one hand it is influenced by the number of largest eigenvalues. On the other hand it is also influenced by the maximum order for the polynomials of the truncated PCEs which describe the uncorrelated random variables inside the truncated KLE. This is exemplified in Table 4.3.

Figure 4.8(a) shows the convergence of the SGM for different truncations of the KLE when the stochastic solution space is spanned by polynomials up to fourth order. The truncations of the KLE satisfy an energy conservation from 99% to 99.999999% (in decimal powers). At this, each of the seven truncated KLEs is represented by the minimum number of eigenvalues required for the specified percentage.

The energy conservation of 99.99% is now fixed for the upcoming numerical experiments (in the next section). Figure 4.8(b) presents the convergence when the maximum order of the stochastic polynomials is increased from the first to the fifth.

p \ energy in %	99	99.9	99.99	99.999
1	10	14	16	17
2	11	19	29	39
3	11	20	30	42
4	11	20	30	42

Table 4.3: *Truncation in the KL representation κ_l: the number of eigenvalues is presented, which is at least required to satisfy an energy conservation from 99% to 99.999%, and its dependency to the maximum order p of the polynomials for the truncated PCEs of the uncorrelated random variables inside the truncated KLE.*

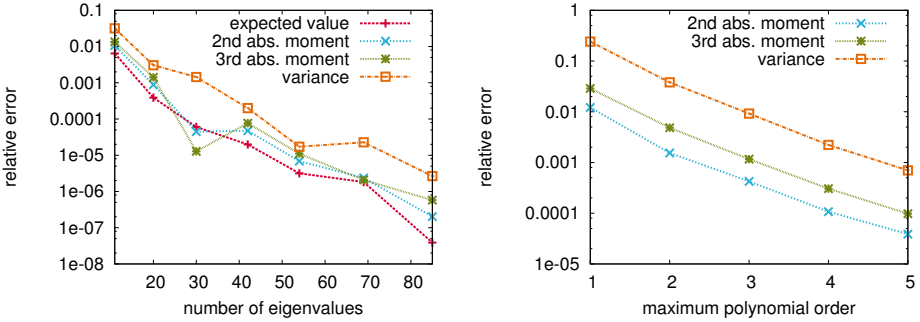


Figure 4.8: *Convergence of the basic SGM with parameter representation κ_l with respect to reduction function $\mathcal{T}_5$ (see Table 4.2): Figure (a) shows the relative error over the number of eigenvalues in the truncated KLE while the maximum order for the polynomials of the stochastic solution space is fixed to four (the references are analytically computed from the direct solution with κ_c for the uncertain parameter); Figure (b) shows the convergence when the maximum order for the polynomials of the solution space is increased from the first to the fifth while κ_l stays fixed at 99.99% of energy conservation (the references are analytically computed from the direct solution based on the same κ_l and basis polynomials up to order six).*

4.1.4 The VLR-SR1U Scheme

The numerical behaviour of the VLR-SR1U scheme in its different configurations (see Section 2.7.2) is demonstrated in this section. The configurations comprise the basic VLR-SR1U scheme, the VLR-SR1U-OPT(1) and VLR-SR1U-OPT(2) schemes, and the VLR-SR1U-ADAPT scheme. The basic VLR-SR1U scheme computes low-rank approximations for the solution. The VLR-SR1U-OPT(1) and VLR-SR1U-OPT(2) schemes optimise the accuracy of already computed low-rank approximations, and the VLR-SR1U-ADAPT scheme constructs the solution space adaptively. The optimisation in VLR-SR1U-OPT(1) and VLR-SR1U-OPT(2) is performed by the VLR-OPT scheme. Independently, the VLR-OPT scheme is applied to optimise the solution obtained by the VLR-SR1U-ADAPT scheme.

The setting of the experiments in this section is specified in the following. The KL representation κ_l (see Table 4.1) with an energy conservation of 99.99% is used to describe the uncertain parameter. Unless otherwise stated the stochastic solution space is spanned by polynomials up to order four. The corresponding set of stochastic polynomials is indicated by $\mathcal{I}^{mod}$ (see Eq. (2.22) in Section 2.1.1.3), which has a cardinality of $|\mathcal{I}^{mod}| = 126$. Reference solutions are obtained through the basic SGM and a direct solving of the corresponding linear system, as well as by an analytical computation of the stochastic moments. As a consequence relative errors are valuable up to machine precision.

The actual task of the VLR-SR1U scheme in practise is to provide low-rank approximations. Nevertheless, the convergence behaviour is here considered up to the full rank where an accuracy of machine precision is possible. This is done to analyse the behaviour of the VLR-SR1U scheme.

An in-depth numerical analysis is carried out in the subsequent sections separately for the different configurations of the VLR-SR1U scheme: the basic VLR-SR1U scheme is discussed in Section 4.1.4.1, the VLR-SR1U-OPT schemes follow in Section 4.1.4.2, and the VLR-SR1U-ADAPT scheme is demonstrated in Section 4.1.4.3. A comparison between the convergence of the VLR-SR1U scheme and the direct integration schemes is focused in Section 4.1.4.4. The results are summarised in Section 4.1.4.5.

4.1.4.1 The Basic VLR-SR1U Scheme

The numerical experiments of the basic VLR-SR1U scheme are presented in this section. Figure 4.9 shows the convergence of relative errors and the residual (more

	tolerance bound					
	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}
sum	683	2,777	5,423	7,275	9,507	12,228

Table 4.4: *The sum of iterations for all rank-one updates up to full rank* 126 *for different tolerance bounds.*

precisely its 2-norm) as well as the process of energy minimisation with respect to successive rank-one updates. Additionally, the number of required iterations within a rank-one update is displayed.

An iteration means to update once the current left vector (for the geometrical domain) and once the current right vector (for the stochastic domain) of the rank-one update. The iteration loop within a rank-one update is beginning in line 4 of the VLR-SR1U algorithm, see Algorithm 2.2 in Section 2.7.2.1. The break criterion of this iteration loop is specified by a maximum number and a tolerance bound. The tolerance bound is set to 10^{-3}. The maximum number is chosen so that the tolerance bound is always breaking the loop (this is done consistently in Section 4.1.4). Obviously, the updated approximations in Figure 4.9 finally reach only an error bound approximately between 10^{-4} and 10^{-6}, depending on the considered stochastic moment. "Finally" means here that the last approximation is of full rank 126. This identifies the approximations as suboptimal. In other words, successive rank-one updates display redundancy in their combined sum. The rank-one update could be of course continued beyond that full rank. However, this is not done here because the full rank is not desirable in practise, it is only covered for analytical reasons. The problem that a current rank approximation is less accurate than possible is addressed by the optimisation scheme VLR-OPT in Section 4.1.4.2.

Next, the influence of the tolerance bound on the quality of the VLR-SR1U scheme is studied. For this purpose the tolerance bound is changed from 10^{-1} to 10^{-6} in decimal powers. Figure 4.10 compares the interesting quantities for the different bounds. An impact of the tolerance bound on the considered relative errors is not noticeable. The convergence of both the residual and the energy is only slightly inferior in the case of the largest bound of 10^{-1}. A clearly huge difference is given by the number of iterations per rank-one update. The sum of iterations for all 126 rank-one updates and a tolerance bound of 10^{-6} is $12,228$. In contrast, the sum of iterations for the bound 10^{-1} is 683, see Table 4.4. One can say that a choice of 10^{-2} may be a good one.

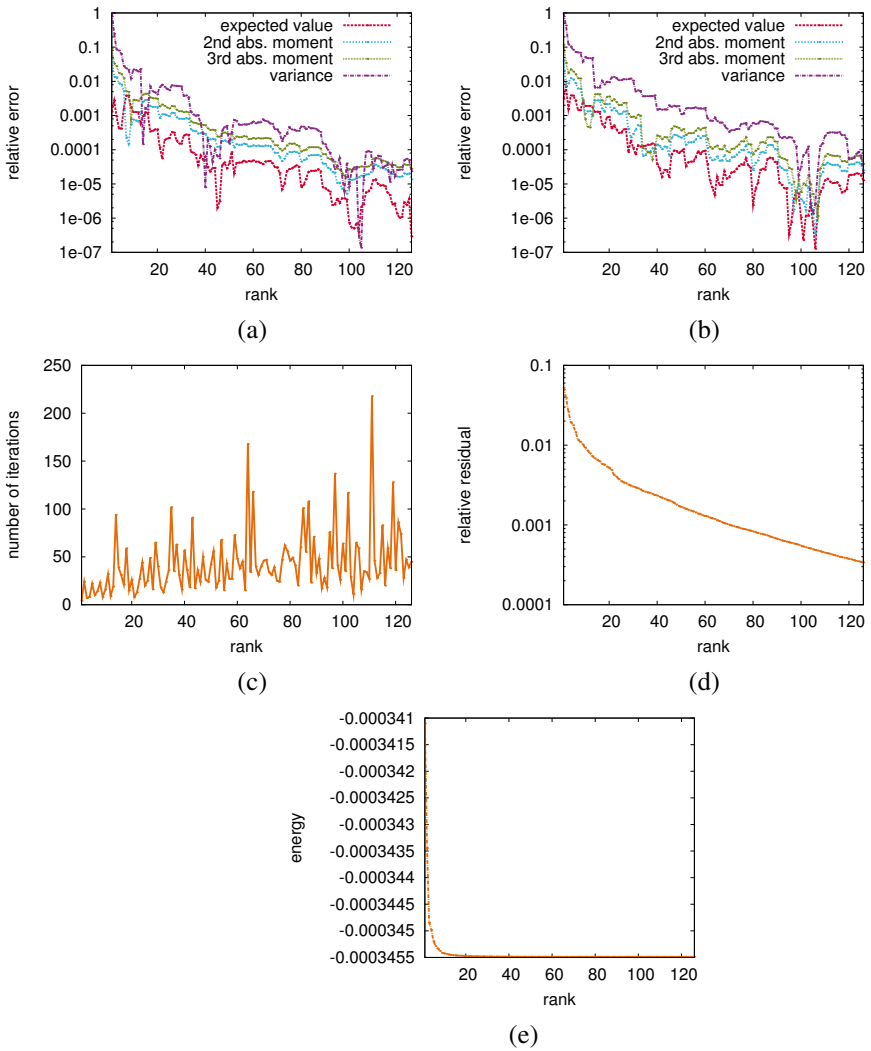


Figure 4.9: *Convergence of the basic VLR-SR1U scheme with tolerance bound* 10^{-3}*: the relative error over the current rank is shown in Figure (a) with respect to* $\mathcal{T}_3$*, and in Figure (b) with respect to* $\mathcal{T}_5$ *(see Table 4.2). The number of iterations, the residual, and the energy are plotted in Figures (c), (d), and (e) over the current rank.*

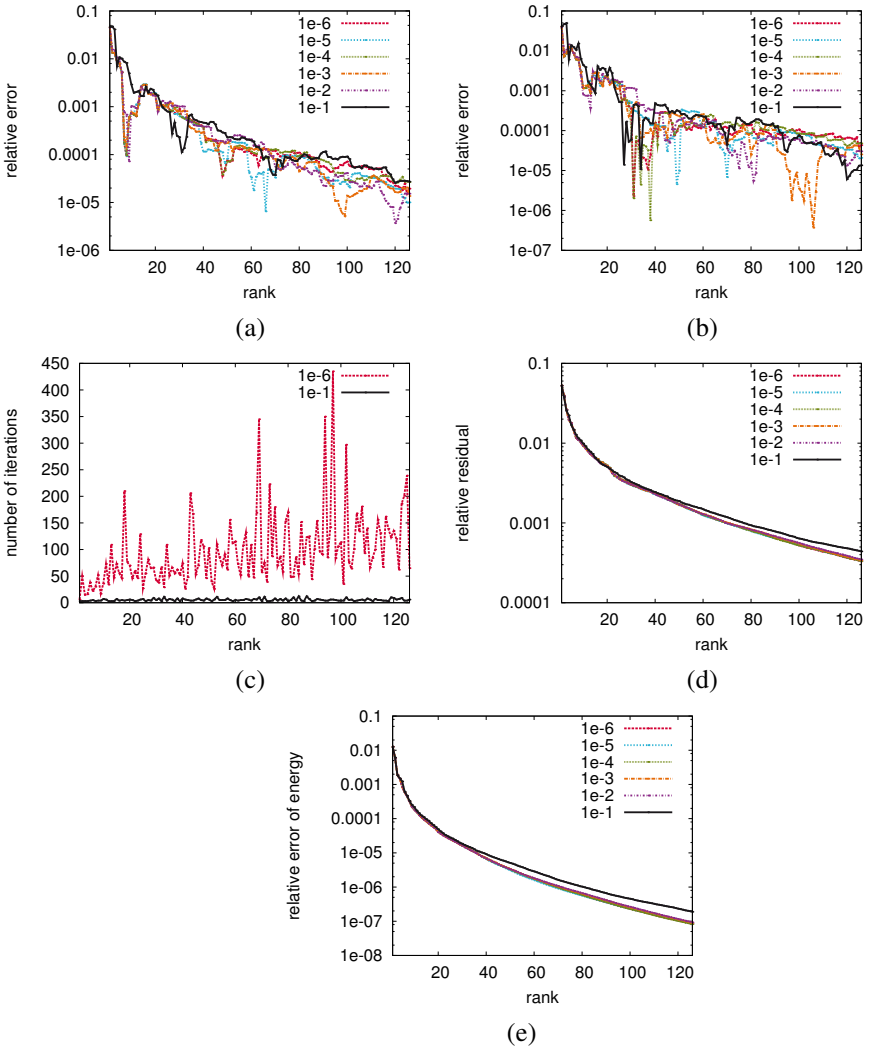


Figure 4.10: *Convergence of the basic VLR-SR1U scheme for tolerance bounds* 10^{-1} – 10^{-6}*: Figures (a) and (b) show the relative error over the current rank for* $\mathcal{T}_3$ *and* $\mathcal{T}_5$*. Figure (c) shows the number of iterations over the current rank for the tolerance bounds* 10^{-1} *and* 10^{-6}*. Figures (d) and (e) show the residual and the relative error of the energy.*

4.1.4.2 The VLR-SR1U-OPT Scheme

The previous section demonstrates that the basic VLR-SR1U scheme provides sub-optimal low-rank approximations. In this section current low-rank approximations are optimised either on the fly — meaning during each rank-one update — by algorithm VLR-SR1U-OPT(1), or in a post-processing step at the end of the preferred rank by algorithm VLR-SR1U-OPT(2) (see Table 2.2 in Section 2.7.2.2).

Figure 4.11 presents the results for both configurations. Figures (a) and (b) compare the VLR-SR1U-OPT(1) scheme with the basic VLR-SR1U scheme. The VLR-SR1U-OPT(2) scheme is compared to the VLR-SR1U-OPT(1) scheme in Figures (c), (d), and (e). The entire number of iterations — occurring in the last three figures — needs a detailed explanation. This is done in the next passage. The setting of the break criteria and a discussion of the observed convergence behaviour are presented afterwards.

One single iteration includes here to update the left vector and the corresponding right vector of a rank-one update. In the case of the basic VLR-SR1U scheme these vectors are $\boldsymbol{g}$ (for the geometrical domain) and $\boldsymbol{h}$ (for the stochastic domain). The corresponding single iteration is located in lines 5 – 8 of the VLR-SR1U algorithm, see Algorithm 2.2 in Section 2.7.2.1. This iteration is named *IT1*. A single iteration of a rank-one update inside the VLR-OPT scheme means either to update the vectors $\boldsymbol{g}$ and $\boldsymbol{v}$ by one iteration, or the vectors $\boldsymbol{h}$ and $\boldsymbol{w}$. These two iterations are located in lines 3 – 6 of Algorithm 2.4 and lines 3 – 6 of Algorithm 2.5 in Section 2.7.2.2 and are referred to as *IT2* and *IT3*. The counter which counts the number of iterations increments by one when IT1, IT2, or IT3 is passed. The three different kinds of iterations are summarised in Table 4.5. In the exterior loop VLR-SR1U additionally iterates over the rank (see line 2 in Algorithm 2.2), and VLR-OPT additionally iterates over the optimisation steps (see line 2 in Algorithm 2.3). In conclusion, the number of iterations inside a complete rank-one update of the VLR-SR1U-OPT(1) scheme contains the number of IT1 iterations, and for each optimisation step the number of IT2 and IT3 iterations. The number of iterations for the VLR-SR1U-OPT(2) scheme comprises the number of IT1 iterations passed by the basic VLR-SR1U scheme to reach the final rank approximation, and the number of IT2 and IT3 iterations inside the subsequent VLR-OPT scheme for each optimisation step.

At first, the comparison between the VLR-SR1U-OPT(1) scheme and the basic VLR-SR1U scheme is discussed. This means to consider Figures 4.11(a) and (b). The used break criteria are defined in the following. The break criterion of the basic VLR-SR1U scheme for each rank-one update is defined by a tolerance of 10^{-3} (the

iteration	algorithm	explanation
IT1	VLR-SR1U	one update of $\boldsymbol{g}$ and $\boldsymbol{h}$
IT2	VLR-OPT	one update of $\boldsymbol{g}$ and $\boldsymbol{v}$
IT3	VLR-OPT	one update of $\boldsymbol{h}$ and $\boldsymbol{w}$

Table 4.5: *Different kinds of iterations in VLR-SR1U configurations (algorithms): the iteration counter increments by one, when IT1, IT2, or IT3 is passed.*

chosen maximum number of iterations is never reached). The VLR-SR1U-OPT(1) scheme is considered in two settings for the break criterion. Both settings contain the same setting for the involved VLR-SR1U scheme: the tolerance of the break criterion is set to 10^{-3} with a maximum of ten iterations. The break criteria for the optimisation steps of the internal VLR-OPT scheme are chosen differently. One setting uses the tolerance of 10^{-4} and a maximum of ten optimisation steps. The corresponding plots in Figures 4.11(a) and (b) show that this setting leads to a better convergence behaviour only up to rank 20. Then the scheme seems to lose its optimisation character, which is actually the case here: the specified tolerance of 10^{-4} is already reached after one optimisation step for all rank approximations after rank 20; consequently, the optimisation is almost not performed. The second setting of VLR-SR1U-OPT(1) uses the tolerance 10^{-15} and the same maximum number of ten optimisation steps. This enforces the passing of all ten optimisation steps in each rank-one update except the first (the rank-one approximation of the involved VLR-SR1U scheme is already of high precision). For this better convergence can be observed, but the initially excellent convergence is slightly less fast for higher ranks. The reason may be that the demand concerning the optimiser VLR-OPT increases for higher ranks. The maximum number of ten is then not enough anymore. This can be confirmed by the discussion of the next figures in the next passage.

Figures 4.11(c), (d), and (e) compare the VLR-SR1U-OPT(2) and VLR-SR1U-OPT(1) schemes. The last mentioned setting with a tolerance of 10^{-15} and a maximum number of ten optimisation steps is used here for representing the VLR-SR1U-OPT(1) scheme. The VLR-SR1U-OPT(2) scheme applies the basic VLR-SR1U scheme with a large tolerance bound of 10^{-1} to quickly obtain an approximation of full rank (the specified maximum number of iterations is never reached). The accuracy of that approximation is then improved in a post-processing step by the involved VLR-OPT scheme. The overall performed $1,250$ optimisation steps finally leads to a precision of approximately 10^{-11}.

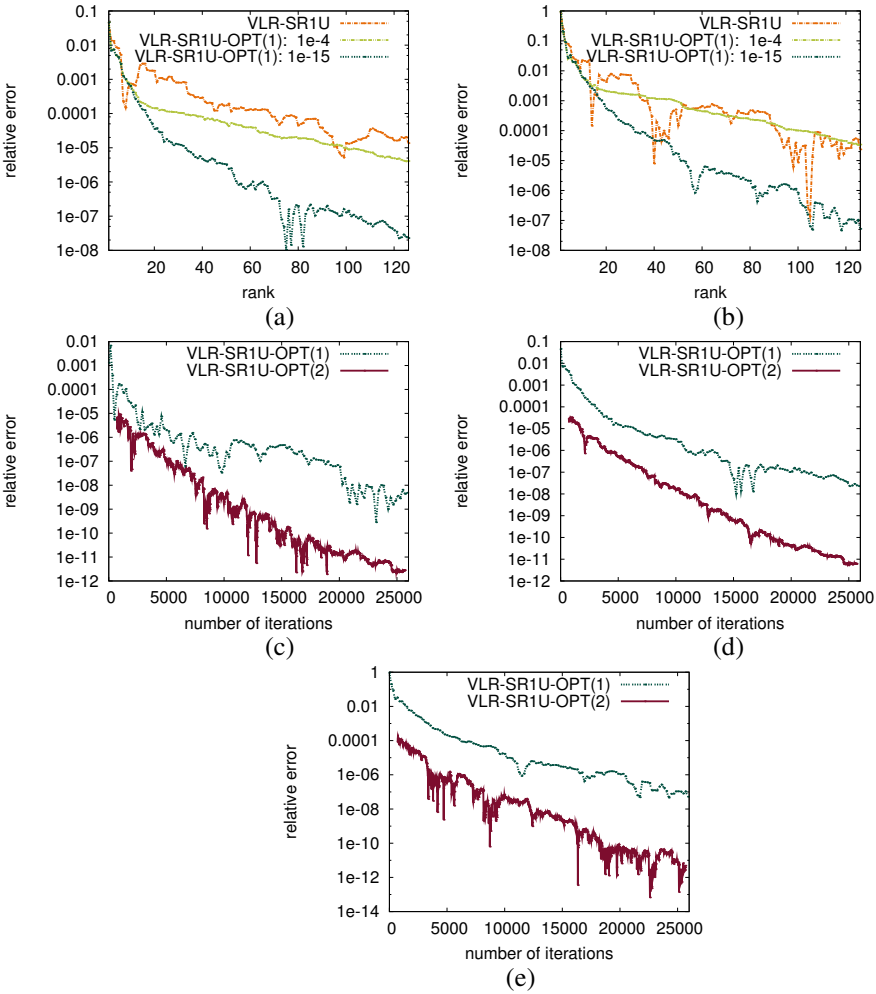


Figure 4.11: *Convergence of VLR-SR1U-OPT: Figures (a) and (b) show the relative error over the rank for the second moment and the variance ($\mathcal{T}_3$). There, VLR-SR1U-OPT(1) is compared to the basic VLR-SR1U (tolerance bound of 10^{-3}). In VLR-SR1U-OPT(1), the tolerance bound 10^{-4} or 10^{-15} is chosen for an optimisation step with up to ten iterations. Figures (c), (d), and (e) show the relative error over the entire number of iterations for the expectation, the second moment, and the variance ($\mathcal{T}_3$). There, VLR-SR1U-OPT(1) with a tolerance bound of 10^{-15} is compared to VLR-SR1U-OPT(2).*

In summary, the VLR-SR1U-OPT(2) scheme shows better convergence than the VLR-SR1U-OPT(1) scheme. Certainly and in contrast to the VLR-SR1U-OPT(1) scheme, the VLR-SR1U-OPT(2) scheme needs to know the final rank a priori.

4.1.4.3 The VLR-SR1U-ADAPT Scheme

The VLR-SR1U-ADAPT scheme constructs the stochastic solution space adaptively and is described by Algorithm 2.7 in Section 2.7.2.3. Its ability to select essential stochastic basis polynomials is demonstrated in this section.

Here, the adaptive construction underlies some chosen properties. The initial stochastic solution space is only spanned by the constant polynomial and is considered by the rank-one approximation. When the rank is incremented by one, a stochastic polynomial is added to the set of current stochastic polynomials. The additional stochastic polynomial is the one which is rated by the involved RBSSE scheme (see Algorithm 2.6) as the best. As a consequence, the resulting VLR-SR1U-ADAPT scheme constructs at rank r a solution space which is spanned by r many stochastic polynomials. Figure 4.12 presents the corresponding convergence experiments. At this, VLR-SR1U-ADAPT is compared to the basic VLR-SR1U scheme. Further settings and the results are discussed in the following passages.

The stochastic polynomials are indicated by the set $\mathcal{I}^{mod}$ (see Eq. (2.22) in Section 2.1.1.3), like in the previous sections. The basic VLR-SR1U scheme uses 56 stochastic polynomials up to third order. In contrast, the VLR-SR1U-ADAPT scheme chooses stochastic polynomials from a set of 126 stochastic polynomials up to fourth order. The references — for the computation of relative errors — are obtained by a direct solution of the corresponding system using a maximum polynomial order of six. The break criteria for the basic VLR-SR1U scheme and the VLR-SR1U-ADAPT scheme are chosen equally: the tolerance bound is set to 10^{-3}, the chosen maximum number of iterations is never reached.

Figures 4.12(a) and (b) plot the convergence of both schemes. The *amount of information of the solution* means the number of floating point variables required to store the low-rank representation of the solution. It is defined as a function of rank r:

$$a(r) := \begin{cases} r(N_X + N_S) & : \text{ fixed solution space} \\ rN_X + \sum_{i=1}^{r} i & : \text{ solution space successively incremented by one.} \end{cases} \tag{4.3}$$

The first case of this definition concerns the low-rank representations provided by the

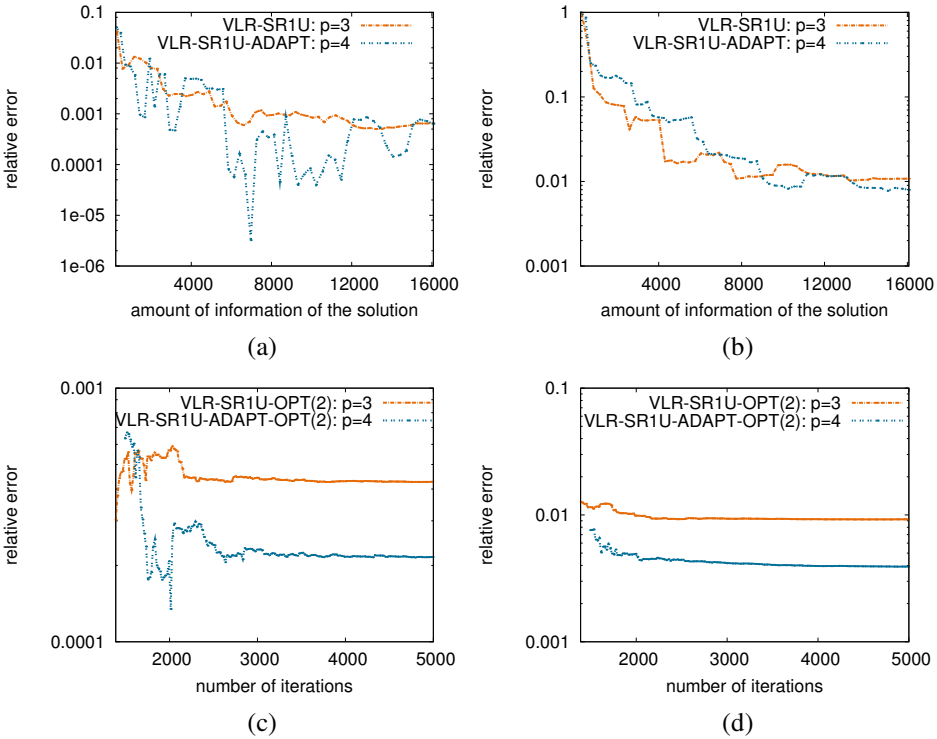


Figure 4.12: *Convergence of the VLR-SR1U-ADAPT scheme: the relative error over the current amount of information (see Eq. (4.3)) with respect to the function $\mathcal{T}_5$ (see Table 4.2) is shown in Figures (a) and (b) for the second absolute moment and the variance. In these figures the basic VLR-SR1U scheme with a stochastic solution space with polynomials up to third order is compared to the VLR-SR1U-ADAPT scheme with a stochastic solution space with polynomials up to fourth order. The references are obtained by the direct solution of the problem using a stochastic solution space with polynomials up to order six. In Figures (c) and (d) the VLR-OPT scheme is applied onto the final rank approximations corresponding to Figures (a) and (b). The schemes — extended by the VLR-OPT scheme — are declared as VLR-SR1U-OPT(2) and VLR-SR1U-ADAPT-OPT(2). The relative error is presented over the current entire number of iterations with respect to function $\mathcal{T}_5$ (see Table 4.2) in Figures (c) and (d) for the second absolute moment and the variance. (p is the maximum polynomial order.)*

basic VLR-SR1U scheme. The second case matches the previously described adaptive construction inside the VLR-SR1U-ADAPT scheme, in which the solution space is extended by one additional stochastic polynomial per rank increment. The VLR-SR1U approximation has the smallest number of involved stochastic basis polynomials, namely 56. As the comparison between the two schemes is done up to the full rank of that approximation the following specification holds: $N_S = 56$ and $r \in \{1, \ldots, N_S\}$.

The convergence in Figures 4.12(a) and (b) is quite similar for the basic VLR-SR1U scheme and the VLR-SR1U-ADAPT scheme. Table 4.6 presents the number of stochastic polynomials at the final rank which share the same order. Obviously, VLR-SR1U-ADAPT also chose stochastic polynomials of the higher — the fourth — order and, in exchange, left out some polynomials of third order. We remember that the approximations of VLR-SR1U are suboptimal. Thus the question arises if the stochastic polynomials of the basic VLR-SR1U scheme or the ones chosen from the VLR-SR1U-ADAPT scheme have the higher potential to describe the solution. For this purpose the rank-56 approximations of both schemes are considered. As already mentioned the solution spaces of both schemes are spanned by 56 stochastic polynomials. The question can be answered by applying the VLR-OPT scheme onto these final approximations. Thus, VLR-OPT is used in a post-processing step. This leads to the already introduced configuration VLR-SR1U-OPT(2) for the basic VLR-SR1U scheme. The combination of VLR-SR1U-ADAPT and VLR-OPT as a post-processor is referred to as the *VLR-SR1U-ADAPT-OPT(2)* scheme. Figures 4.12(c) and (d) show the relative error over the entire number of iterations (see Section 4.1.4.2 for the definition). The figures indicate the advantage of the VLR-SR1U-ADAPT scheme: the chosen set of 56 stochastic polynomials describes the solution more accurate than the set corresponding to the basic VLR-SR1U approximation.

	order of stochastic basis polynomials				
	0	1	2	3	4
basic VLR-SR1U	1	5	15	35	0
VLR-SR1U-ADAPT	1	5	15	26	9

Table 4.6: *The numbers of stochastic basis polynomials at the final rank, categorised through their order: the basic VLR-SR1U scheme operates on a stochastic solution space spanned by polynomials up to third order, the VLR-SR1U-ADAPT scheme also selected stochastic basis polynomials of fourth order.*

4.1.4.4 The VLR-SR1U Scheme versus Direct Integration Schemes

The convergence of the VLR-SR1U scheme and the direct integration schemes are compared in this section. It can be observed in the following that the VLR-SR1U scheme operates slightly worse than the most efficient direct integration scheme, namely the Smolyak algorithm with the one-dimensional Kronrod-Patterson quadrature rule (abbreviated by S-KP).

A common validation for the convergence of direct integration schemes answers the question how many samples of the probabilistic model need to be simulated to reach a specific relative error. This validation can also be applied for the VLR-SR1U scheme. Each iteration inside the basic VLR-SR1U scheme or each iteration inside an optimisation step of the VLR-SR1U-OPT schemes indicates one of these simulations.

The basic VLR-SR1U scheme with a tolerance bound of 10^{-1} and 10^{-3} (see Section 4.1.4.1) and the VLR-SR1U-OPT(1) scheme with a tolerance bound of 10^{-4} are considered. The latter is chosen because it performs well during the first rank-one updates, see Section 4.1.4.2; the suboptimal behaviour at higher ranks is not visible here because the limits for the validity of relative errors are too large (see next passage). Figure 4.13 presents appropriate convergence plots for the expectation and the third absolute moment.

The relative errors are computed with respect to the MC references. On the one hand these references are only valuable up to the stated accuracies. On the other hand the stochastic solution space spanned by polynomials up to fourth order and the KL representation of the uncertain parameter limits the accuracy which is reachable for the VLR-SR1U scheme. The limit for the relative error of the expectation, for which a consideration is valuable, is declared by the MC reference; the one for the relative error of the third absolute moment is declared by the result of the SGM in Section 4.1.3 for the mentioned stochastic solution space and the KL representation of the uncertain parameter with an energy conservation of 99.99%.

The most efficient configuration for the VLR-SR1U scheme is the basic one with a tolerance bound of 10^{-1}. This is not surprising. It is observed in Section 4.1.4.1 that on the one hand the basic VLR-SR1U scheme performs well especially at the first ranks, and on the other hand its accuracy is almost invariant with respect to the tolerance bound. As the tolerance bound of 10^{-1} requires very few iterations per rank increase, the number of required samples is also small. Nevertheless, that configuration of the basic VLR-SR1U scheme quickly reaches higher ranks: each plotted dot

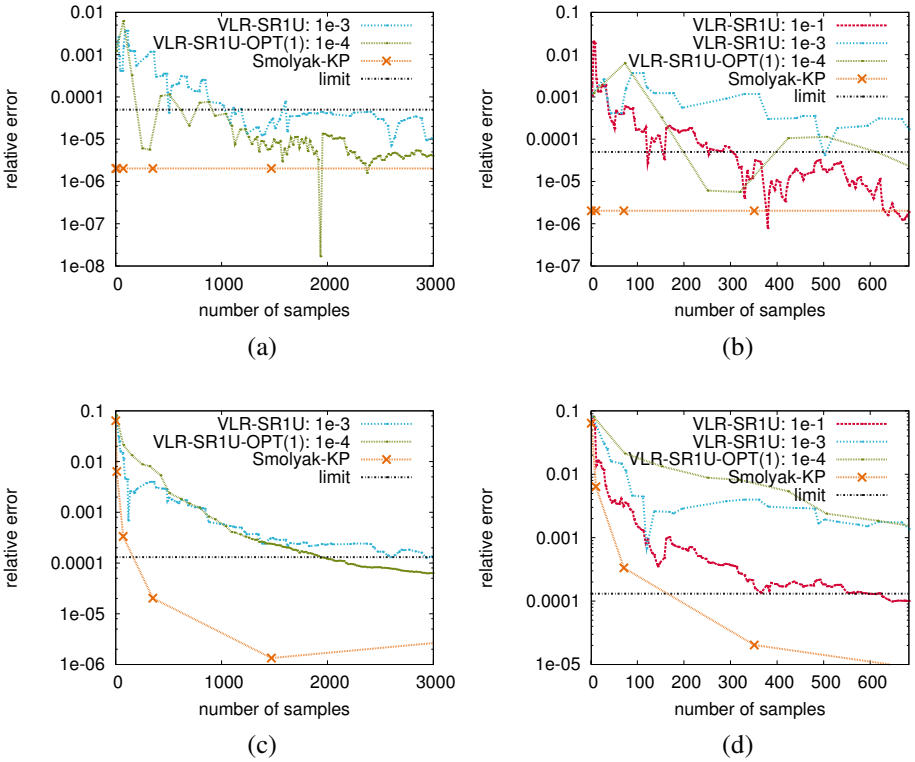


Figure 4.13: *Comparison between the convergence of the VLR-SR1U scheme and the Smolyak algorithm (with the one-dimensional Kronrod-Patterson quadrature rule) for the reduction function $\mathcal{T}_1$: the convergence with respect to the expected value is shown in Figures (a) and (b), at which Figure (b) also displays the best configuration for the VLR-SR1U scheme; analogously, Figures (c) and (d) show the convergence with respect to the third absolute moment.*

of the corresponding (linearly interpolated) convergence graph in Figure 4.13 means a rank increase for the VLR-SR1U scheme. In contrast, the VLR-SR1U-OPT(1) scheme requires more samples of the probabilistic model but keeps the rank low.

Obviously, the VLR-SR1U scheme cannot compete with S-KP with respect to the

convergence of the expectation, see Figures 4.13(a) and (b). However, the basic VLR-SR1U scheme with tolerance bound 10^{-1} already reaches the limit of accuracy at 304 samples of the probabilistic model, while the QMC method based on the Halton point sequence requires around $50,000$ samples (see Figure 4.4(b) in Section 4.1.2), and the basic MC method requires some millions of samples (see Figure 4.3(b) in Section 4.1.2). The convergence of the basic VLR-SR1U scheme with tolerance bound 10^{-1} is a bit inferior than the one of S-KP for the third stochastic moment, see Figures 4.13(c) and (d): the VLR-SR1U scheme requires 625 samples in contrast to 351 samples for S-KP and $100,000$ samples for the QMC method.

4.1.4.5 Conclusion

The numerical behaviour of the VLR-SR1U scheme in its different configurations is discussed in the previous sections. A compact overview about the corresponding results is provided in this section.

The approximations of the basic VLR-SR1U scheme are suboptimal: at the full rank the bounds of the relative errors are between 10^{-4} and 10^{-6}, depending on the considered quantity, see Section 4.1.4.1. This problem is addressed by an optimisation of a given low-rank approximation through the VLR-OPT scheme. The VLR-SR1U-OPT(1) scheme uses VLR-OPT during each rank-one update. In this way the mentioned error bound is decreased. It may decrease down to machine precision, but this depends on the setting of the break criterion for the optimisation loop, see Section 4.1.4.2. The VLR-SR1U-OPT(2) scheme applies VLR-OPT only at the final rank: in comparison to VLR-SR1U-OPT(1) a better convergence can be observed, see Section 4.1.4.2. The disadvantage of VLR-SR1U-OPT(2) is the requirement of an a priori specification of the final rank.

The VLR-SR1U-ADAPT scheme extends the basic VLR-SR1U scheme to an adaptive construction of the solution space. While the stochastic basis polynomials for the basic VLR-SR1U scheme were limited by an order of three, the VLR-SR1U-ADAPT scheme could also select stochastic polynomials of fourth order. For an approximation of a fixed rank it was observed that VLR-SR1U-ADAPT chose stochastic polynomials of fourth order by leaving out some of third order. However, the basic VLR-SR1U scheme and the VLR-SR1U-ADAPT scheme showed similar convergence. Then, the VLR-OPT scheme was applied on both approximations. In this way it could be shown that the adaptively chosen stochastic polynomials were

describing the solution more accurately than the a priori fixed ones from the basic VLR-SR1U scheme, see Section 4.1.4.3.

A comparison between the convergence of the VLR-SR1U scheme and direct integration schemes shows in Section 4.1.4.4 that the VLR-SR1U scheme is a bit inferior to the best direct integration scheme applied here, namely the Smolyak algorithm with the one-dimensional Kronrod-Patterson quadrature rule. (Q)MC methods are far behind. The VLR-SR1U scheme cannot compete with that configuration of the Smolyak algorithm concerning the convergence for the expectation, but only few hundreds of samples are required, while the QMC and MC methods require ten thousands and millions of samples.

4.1.5 Conclusion

Direct integration schemes and the VLR-SR1U scheme in its different configurations are applied in this section to simulate a two-dimensional stationary groundwater flow problem with an uncertain hydraulic conductivity. The convergence of these numerical schemes is compared.

The applied direct integration schemes are a basic MC method, a QMC method based on the Halton point sequence, and the Smolyak algorithm. Different kinds of one-dimensional quadrature rules are chosen for the Smolyak algorithm. The best convergence in the segment of the direct integration schemes is provided by the Smolyak algorithm with the one-dimensional Kronrod-Patterson quadrature rule (S-KP). While this configuration of the Smolyak algorithm requires only a few hundred samples for higher moments, the QMC or MC method requires ten thousands or millions of samples to reach the same accuracy.

It can be demonstrated that the VLR-SR1U scheme provides slightly inferior convergence than S-KP, but it is far better than (Q)MC methods. Furthermore, the VLR-SR1U-OPT(1) and VLR-SR1U-OPT(2) schemes can be successfully applied to optimise a low-rank approximation with respect to the expectation of the total potential energy so that higher accuracies can be reached. The VLR-SR1U-ADAPT scheme is used to construct the solution space adaptively. It is shown that the constructed solution space leads to higher accuracies than a non-adaptively constructed one when stochastic polynomials of higher order can be chosen by the VLR-SR1U-ADAPT scheme.

4.2 A Laminated Composite Material

This section addresses the modelling of a fully anisotropic and uncertain carbon-fibre-reinforced composite material of a three-dimensional rigid body. The model is constructed from statistics which are obtained through the processing of photographic images of a three-dimensional test specimen which is not externally loaded. Here, the statistics are the input, nevertheless the process to obtain them is shortly discussed; a corresponding overview is given in [94].

The model for the composite material is divided into uncertain interlayers. Each interlayer is separately considered by its own model and described through a non-Gaussian random field, which is defined on a vector of standard Gaussian random variables. The first obtained representation of the random field is simply called the "generator". Further representations are derived on the basis of this generator, namely a PCE and a KLE. These three representations are numerically constructed for a selected interlayer from the photographic input. Six hundred Gaussian random variables are required to get an adequate description for the interlayer. This comparatively large stochastic dimension causes numerical difficulties and integration, projection, and regression techniques are applied to handle them.

The exemplarily constructed numerical model for the selected uncertain interlayer is used to describe the material of a laminated composite structure with a linear constitutive law. The structure is simulated by different numerical schemes. A reference solution is obtained by a basic MC method. The basic VLR-SR1U and the VLR-OPT schemes are applied to search for an accurate low-rank approximation of the solution in a given solution space. It turns out that, at this, a relatively high rank is required. The low-rank approximation is compared to the solution of a regression approach, which is determined in the same solution space.

The carbon-fibre-reinforced composite material and especially its modelling and numerical handling are explained in Section 4.2.1. The mentioned linear problem of structural mechanics and its simulation are discussed in Section 4.2.2.

The experiments are related to the project "More Affordable Aircraft through eXtended, Integrated and Mature nUmerical Sizing" (MAAXIMUS) — see `http://www.maaximus.eu` — funded by the European Community's Seventh Framework Programme FP7/2007–2013 under grant agreement $n^o 213371$. (The project is concerned with the simulation of a loaded three-dimensional clipping of a fuselage for which the naturally inherent uncertainty of the composite material is stochastically described.) The contributions of this thesis are, on the one hand, an active partici-

pation in the last conceptual phase of the material modelling and the execution and preparation of the corresponding numerical computations (this is additionally clarified in the following section) and, on the other hand, the simulation of the laminated composite structure with a linear constitutive law.

4.2.1 A Model for a Carbon-Fibre-Reinforced Composite Material

A laminated carbon-fibre-reinforced composite material consists of glued layers of carbon fibres. The orientation of the fibres within a layer is identical, but the fibre orientation usually changes from layer to layer. A carbon fibre has a diameter of around 5–8 micrometres [38]. The adhesive — the so-called "matrix" — builds intermediate layers between the layers of carbon fibres. It includes air, referred to as "void", which disrupts the contact between the layers. In addition it can be observed that the borders between layers are irregular.

The composite material is divided into virtual interlayers. These interlayers are introduced to especially describe the uncertain characteristics which are induced by the mentioned voids and irregular borders. The fully anisotropic and uncertain local material tensor for the composite material is represented by a random field $\boldsymbol{\kappa} : \mathbb{R}^3 \times \Omega \to \mathrm{Sym}(\mathbb{R}^{6\times 6})$, and determined for the interlayers separately. $\boldsymbol{\kappa}$ is specified by 21 random variables (instead of 36) due to the symmetric structure of the tensor.

The interlayers are described in detail in Section 4.2.1.1. The numerical model for an interlayer is constructed in Section 4.2.1.2, and corresponding numerical experiments are discussed in Section 4.2.1.3. A conclusion is presented in Section 4.2.1.4.

4.2.1.1 The Interlayers of the Composite Material

Photographic images of a test specimen are the input of the composite material model and are provided by a cooperation partner. The test specimen has a size of $s_B = (l_B, w_B, h_B) = (14.875, 14.875, 4.675)$ with millimetres as the unit of length. It is grinded and the grinded surface is photographed at the standardised grinding depth of $d_g = 100$ micrometres. This procedure is repeated with the rest of the test specimen as many times as possible to obtain a sequence of depth images, see the drawing in Figure 4.14. The drawing illustrates the physical layered structure of the composite

material and identifies 63 layers, namely 32 layers of carbon fibres and 31 layers of adhesive.

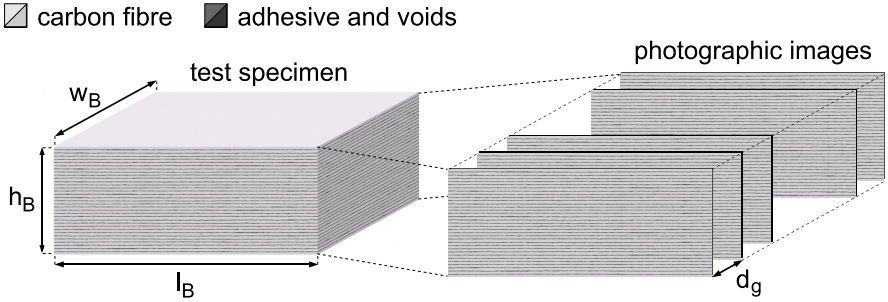


Figure 4.14: *Systematic depiction of the test specimen of size s_B on the left and a sequence of photographic images resulting from a grinding process on the right.*

The layers of carbon fibres can be categorised according to their fibre orientation. Figure 4.15 shows a section of two glued layers of carbon fibres with the adhesive layer in between. The size of the section measures $s_{\text{RVE}} = (l_{\text{RVE}}, w_{\text{RVE}}, h_{\text{RVE}}) = (100, 100, 100)$ with a measurement unit of micrometres for each entry; this size is used later in the modelling process for a representative volume element (*RVE*). Additionally, the depiction shows the already mentioned irregular borders between the layers.

A virtual layered structure is introduced to capture the uncertainty, which is generated by voids and irregular borders. The relation between the physical layered structure and the virtual one is illustrated in Figure 4.16. The virtual layered structure contains two kinds of layers: the interlayer and the deterministic layer. An interlayer consists of a complete physical adhesive layer plus areas of the neighbouring layers of carbon fibres above and below that adhesive layer. Accordingly, the interlayers altogether comprise the specified uncertain characteristics. An interlayer has a height of $h_{inter} = h_{\text{RVE}}$. A deterministic layer contains the remaining area of the carbon fibres either between the top (or bottom) of the test specimen and the first (or last) interlayer, or it contains the remaining area of the carbon fibres between two neighbouring interlayers. Accordingly, the set of all deterministic layers comprises the remaining area of carbon fibres; a deterministic layer has a height of $h_{det} = 49.21875$ micrometres. Therefore the numbers of physical layers and virtual layers are the same.

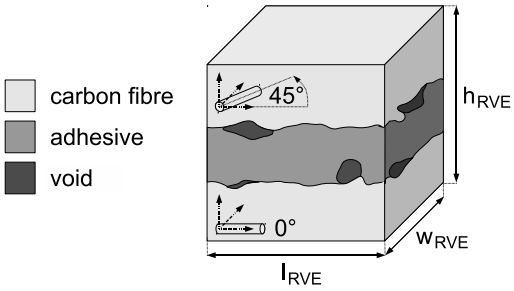


Figure 4.15: *Systematic depiction of a section of two glued layers of carbon fibres. The upper layer exhibits a fibre orientation of* 45 *degrees; the lower layer exhibits a fibre orientation of* 0 *degrees.*

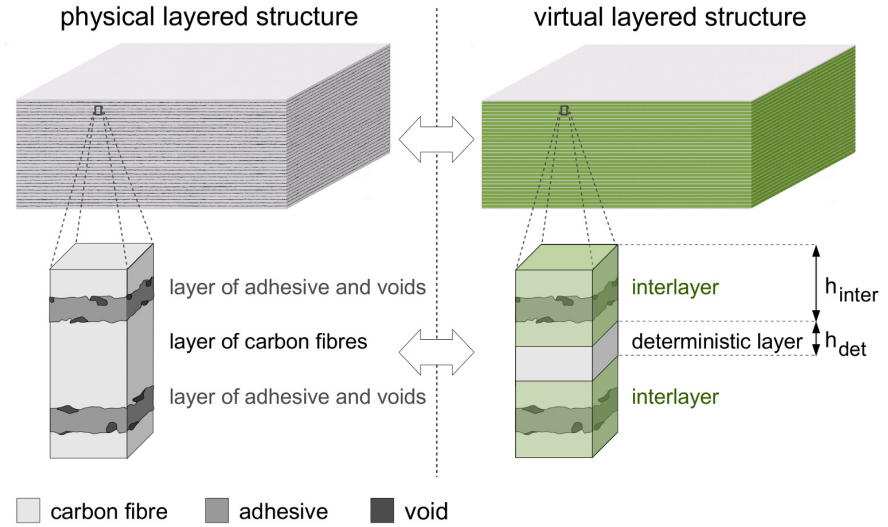


Figure 4.16: *The relation between the physical (inherent) layered structure and the virtual layered structure of the test specimen.*

Each virtual layer is geometrically discretised by a plane of hexahedral finite elements with the height of one finite element. Thus the height of the finite elements makes up the height of the corresponding layer. The material input for the finite elements of an interlayer is defined onto a two-dimensional plane, which is located in the centre of the FE plane. This two-dimensional plane is called the *reference plane*. Figure 4.17 exemplifies the discretisation for some successive virtual layers.

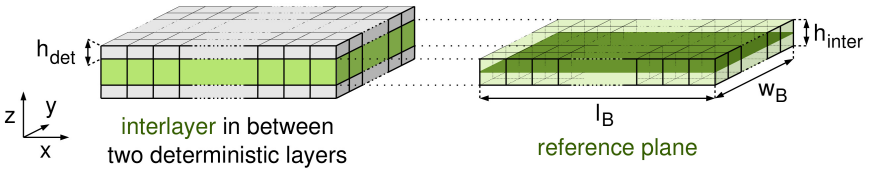


Figure 4.17: *The concept for the discretisation of the virtual layers of the composite material by hexahedral finite elements: an interlayer in between two deterministic layers is shown on the left; the reference plane of the interlayer is shown on the right (it marks the material input for the finite elements of the interlayer).*

In summary, the model of an interlayer goes back to the computation of the material defined onto the related reference plane. Section 4.2.1.2 explains the concepts for computing this material.

4.2.1.2 A Model for an Interlayer

The reference plane of the k'th interlayer is given by the set $X_k := \{(x, y) \mid (x, y, z_k) \in X_B\}$, at which $X_B := [0, \ldots, l_B] \times [0, \ldots, w_B] \times [0, \ldots, h_B]$ is the three-dimensional domain of the test specimen, and z_k is the constant z-coordinate of the two-dimensional reference plane. The random field $\boldsymbol{\kappa}_k : X_k \times \Omega \to \mathrm{Sym}(\mathbb{R}^{6\times 6})$ is introduced to describe the material defined onto that plane. The aim is to construct concrete representations for $\boldsymbol{\kappa}_k$ from the given photographs.

The construction can be divided in two phases. In the first phase the photo samples are processed to obtain a stochastic description for the interlayer in the form of marginal distributions and covariance functions. In the second phase these stochastic quantities are used to construct a generator $\boldsymbol{g}_k$, which is already the first representation for $\boldsymbol{\kappa}_k$. An elementary event $\omega \in \Omega$ exclusively affects an m-dimensional

standard Gaussian random vector $\boldsymbol{\theta}$ in $\boldsymbol{g}_k$ so that the notation $\boldsymbol{g}_k((x,y),\boldsymbol{\theta})$ — instead of $\boldsymbol{g}_k((x,y),\omega)$ — can also be used. Further representations for $\boldsymbol{\kappa}_k$ are a PCE $\boldsymbol{\kappa}_{k,c}$ and a KLE $\boldsymbol{\kappa}_{k,l}$ (c and l indicate truncations). Both representations are based on a sampling of $\boldsymbol{g}_k$. As a consequence of the truncations, $\boldsymbol{g}_k$ carries the most information of the input compared to the other concrete representations.

First Phase The first phase in the construction of a material model for an interlayer is not a contribution of this thesis. Therefore, that phase is merely outlined in this paragraph and essential numerical results are mentioned. An overview of the first phase is published in [94], further publications are in preparation. As the photographs were only available for a single test specimen an uncertainty quantification for each location-dependent pixel could not be accomplished. Therefore, a RVE is introduced, which represents the uncertainty within its geometrical volume. The size s_{RVE} of the RVE matches the grinding depth d_g in y-direction and the interlayer height h_{inter} in z-direction. A scan-element of size s_{RVE} is now pixel-wise moved from one end of the k'th interlayer to the other end, see Figure 4.18. Two photos of consecutive grinding depths are read to get the front and back of the scanned volume. (An interpolation is later performed for the space in between the front and back.) The scanning is done for each pair of photos corresponding to consecutive grinding depths. Consequently, plenty of data are obtained for the uncertainty quantification, namely $170,000$ samples of the RVE. A model reduction and a quantification of the resulting stochastic coefficients lead to $3,072$ samples, for which the local material tensors are identified through load tests of RVE-sized models of structural mechanics. A surrogate model is obtained from these $3,072$ samples, with which the matrix logarithms of the local material tensors for all $170,000$ samples can be approximated. The matrix logarithm is introduced to finally assure a positive-definite material representation. Further model reduction for all these matrix logarithms and a quantification of the resulting stochastic coefficients lead to two uncorrelated random variables s_0 and s_1. As a consequence, the function

$$\varsigma : \mathbb{R}^2 \to \text{Sym}(\mathbb{R}^{6\times 6}) \tag{4.4}$$

is established, which maps a sample of s_0 and s_1 to the matrix logarithm $\boldsymbol{L}_k = \log(\boldsymbol{K}_k)$, at which $\boldsymbol{K}_k$ is the corresponding sample of the local material tensor for the k'th interlayer. The matrix exponentiation can be used to obtain $\boldsymbol{K}_k$: $\boldsymbol{K}_k := \exp(\boldsymbol{L}_k)$. It assures the positive-definiteness of the local material tensor. (If the previous model reduction had been directly performed on the local material tensors the positive-definiteness could not have been assured for a final material representation.) However, s_0 and s_1 are not independent. As independence simplifies the second phase presented in the next paragraph, the Rosenblatt transformation τ [182]

is applied to transform s_0 and s_1 to two independent standard Gaussian random variables r_0 and r_1:

$$\tau : (s_0, s_1) \mapsto (r_0, r_1). \tag{4.5}$$

As a consequence a function ζ can be defined, which maps a sample of the Rosenblatt variables onto the corresponding local material tensor:

$$\zeta(r_0, r_1) := \exp(\varsigma(\tau^{-1}(r_0, r_1))). \tag{4.6}$$

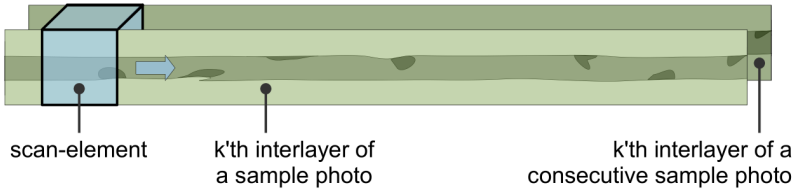


Figure 4.18: *The concept for scanning an interlayer: the scan-element is pixel-wise moved over two consecutive photo samples.*

The marginal distributions F_{r_0} and F_{r_1} and the covariance functions c_{r_0} and c_{r_1} of r_0 and r_1 are now extracted for the next phase. The numerical experiments identify the Rosenblatt variables to be only almost standard Gaussian.

Second Phase In the second phase concrete representations for $\boldsymbol{\kappa}_k$ are constructed, namely the generator $\boldsymbol{g}_k$ and the truncated PCE and KLE $\boldsymbol{\kappa}_{k,c}$ and $\boldsymbol{\kappa}_{k,l}$. For this purpose, the random field representations ρ_0 and ρ_1 are determined, which hold the Rosenblatt quantities F_{r_0}, F_{r_1}, c_{r_0}, and c_{r_1}. The function ζ in Eq. (4.6) is then used to construct $\boldsymbol{g}_k$. The entire process is discussed in the following and illustrated by Figure 4.19.

As r_0 and r_1 are independent, so are ρ_0 and ρ_1 and their corresponding computations. ρ_0 and ρ_1 are non-Gaussian random fields, as r_0 and r_1 are only almost Gaussian. The processes to obtain ρ_0 and ρ_1 are identical so that only one process needs to be exemplified. The algorithm in [185] is applied as follows to obtain a truncated PC representation $\rho_{0,c}$ for ρ_0 from the input F_{r_0} and c_{r_0} (analogously for ρ_1). The

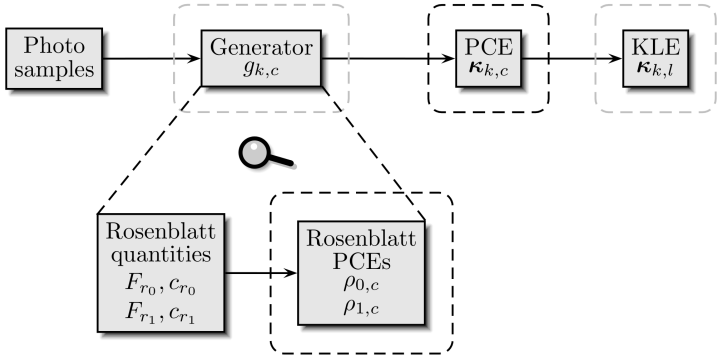


Figure 4.19: *The process to determine representations for κ_k from photo samples: the black dashed lines mark the contributions of this thesis; the grey dashed lines mark contributions jointly achieved with a cooperation partner.*

Rosenblatt variable r_0 is projected onto a set of one-dimensional Hermite polynomials $\{h_i\}_{i\in\{0,\dots,p\}}$ defined on a standard Gaussian random variable θ:

$$r_0 \approx r_{0,p} := \sum_{i=0}^{p} r_0^{(i)} h_i(\theta). \tag{4.7}$$

p is the polynomial order of the projection. The computation of the coefficients $\{r_0^{(i)}\}_{i\in\{0,\dots,p\}}$ involves F_{r_0}. Analogously, ρ_0 can be considered as a projection onto the same Hermite polynomials but defined on a centred Gaussian field γ_0 with unit variance [185]. Then $\{r_0^{(i)}\}_{i\in\{0,\dots,p\}}$ is also the set of coefficients for that projection:

$$\rho_0 \approx \sum_{i=0}^{p} r_0^{(i)} h_i(\gamma_0). \tag{4.8}$$

When one follows this analogy a representation for γ_0 needs to be found. The transformation from covariance function c_{r_0} of ρ_0 to the correlation function c_{γ_0} of γ_0 can be expressed by the coefficients $\{r_0^{(i)}\}_{i\in\{0,\dots,p\}}$. The corresponding problem can be identified by the problem to find the roots of an associated polynomial of order p [185]. Afterwards, c_{γ_0} is used to compute a truncated KL representation γ_{0,m_0} for γ_0, for which m_0 is the number of involved standard Gaussian random variables. γ_{0,m_0} substitutes γ_0 in Eq. (4.8) to get an approximation for ρ_0. This approximation is projected onto a set of Hermite polynomials $\{H_\alpha\}_{\alpha\in\mathcal{I}^c}$, which are defined on

the m_0-dimensional standard Gaussian random vector $\boldsymbol{\theta}_0$ to obtain a truncated PC representation $\rho_{0,c}$ for ρ_0 [185].

The random fields ρ_0 and ρ_1 can be used to construct the already introduced generator $\boldsymbol{g}_k$ for the local material tensors: $\boldsymbol{g}_k((x,y),\omega) = \zeta(\rho_0((x,y),\omega),\rho_1((x,y),\omega))$. Here the two computed PC representations $\rho_{0,c}$ and $\rho_{1,c}$ are replacing ρ_0 and ρ_1 for a practical application. The resulting $m_0 + m_1 = m$-dimensional generator is defined by

$$\boldsymbol{g}_{k,c}\big((x,y),\boldsymbol{\theta}\big) := \zeta\Big(\rho_{0,c}\big((x,y),\boldsymbol{\theta}_0\big),\rho_{1,c}\big((x,y),\boldsymbol{\theta}_1\big)\Big) \tag{4.9}$$

with $\boldsymbol{\theta} := (\boldsymbol{\theta}_0,\boldsymbol{\theta}_1)$. The generator $\boldsymbol{g}_{k,c}$ may already be applied in sampling schemes like (Q)MC methods, the Smolyak algorithm, and projection or stochastic collocation methods. However, $\boldsymbol{g}_{k,c}$ is sampled to obtain a truncated PC representation $\boldsymbol{\kappa}_{k,c}$ for $\boldsymbol{\kappa}_k$. Subsequently, a truncated KL representation $\boldsymbol{\kappa}_{k,l}$ is computed by using $\boldsymbol{\kappa}_{k,c}$.

4.2.1.3 Numerical Experiments Concerning an Interlayer

The numerical experiments presented in this section exclusively focus on the second phase of modelling an interlayer, and are a contribution of this thesis. Essential numerical results of the first phase were already presented in Section 4.2.1.2.

The second phase determines approximations for $\boldsymbol{\kappa}_k$ from the quantities F_{r_0}, F_{r_1}, c_{r_0}, and c_{r_1}, corresponding to the Rosenblatt fields ρ_0 and ρ_1. The associated process is outlined in Figure 4.19 of Section 4.2.1.2. In this section, the third interlayer is considered, that means $k = 3$. The interlayer exhibits an upper fibre orientation of -45 degrees and a lower fibre orientation of 0 degrees. The Rosenblatt input for the second phase is presented in Figure 4.20: the covariance functions are demonstrated along the x- and y-directions of the reference plane and along the diagonal direction between the x- and y-direction with a length of $\sqrt{(l_B)^2 + (w_B)^2}$. The geometrical domain X_k of the reference plane is discretised by $N_n := 51 \times 51 = 2,601$ nodes for further numerical experiments. Unless otherwise stated the set of multi-indices in Eq. (2.22) is used for representing multi-dimensional Hermite polynomials.

Truncated PC Representations for the Rosenblatt Quantities First of all, truncated PC representations $\rho_{0,c}$ and $\rho_{1,c}$ for ρ_0 and ρ_1 are computed as explained in Section 4.2.1.2. This includes the computation of the coefficients $\{r_0^{(i)}\}_{i\in\{0,\dots,p\}}$ and

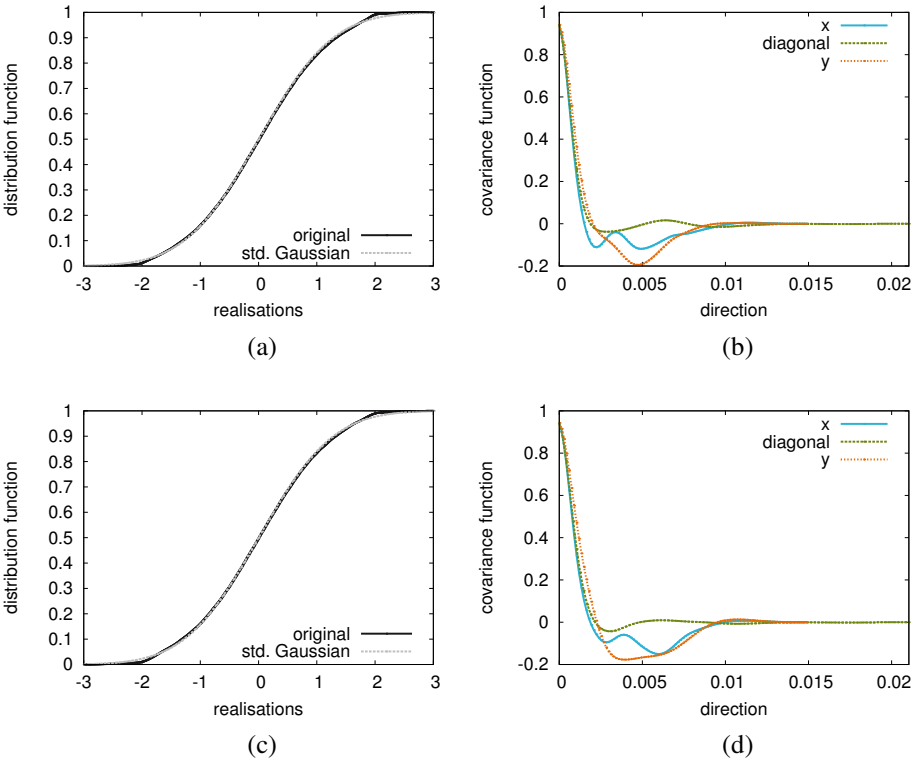
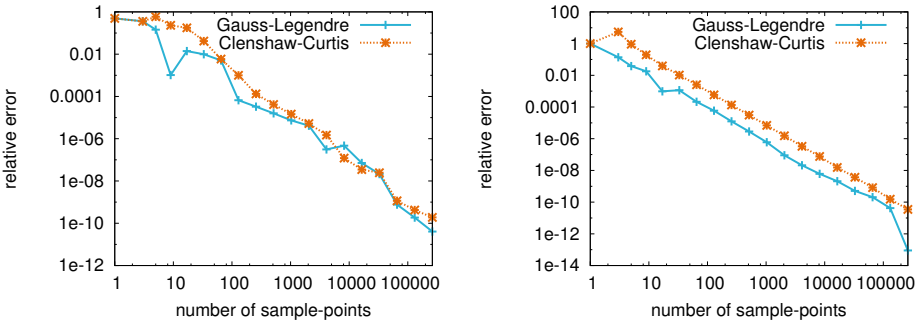


Figure 4.20: *The Rosenblatt input for the third interlayer: the marginal distribution and the covariance function of Rosenblatt field ρ_0 are shown in Figures (a) and (b); these quantities are analogously shown for Rosenblatt field ρ_1 in Figures (c) and (d). The particular covariance function is demonstrated along the x- and y-direction and the diagonal direction of the corresponding reference plane.*

$\{r_1^{(i)}\}_{i\in\{0,\dots,p\}}$ corresponding to the projections $r_{0,p}$ and $r_{1,p}$ of the Rosenblatt variables r_0 and r_1 onto a set of one-dimensional Hermite polynomials. The associated process involves the approximation of an integral for each coefficient (analogously to Eq. (2.6)). For this purpose, the Gauss-Legendre (*GL*) and Clenshaw-Curtis (*CC*) quadrature rules (see Section 2.3.2.1) are applied and compared with respect to their convergence. Figure 4.21 exemplifies the convergence of both quadrature rules for coefficients $r_0^{(0)}$ and $r_0^{(1)}$; the convergence is almost the same with a slight vantage

for GL, especially at the beginning. Figure 4.22 exemplifies the convergence of the distribution function of $r_{0,p}$ to F_{r_0} by increasing the maximum polynomial order p. Based on these experiments GL quadrature with $500,000$ sample-points and a maximum polynomial order of $p = 30$ are used to obtain accurate approximations $r_{0,p}$ and $r_{1,p}$ for r_0 and r_1.



(a) (b)

Figure 4.21: *Convergence of Gauss-Legendre and Clenshaw-Curtis quadrature rules for the coefficient $r_0^{(0)}$ in Figure (a) or $r_0^{(1)}$ in Figure (b). A reference for the relative error is obtained by Gauss-Legendre quadrature with $1,100,000$ sample-points.*

Next, the truncated KL representations γ_{0,m_0} and γ_{1,m_1} for the Gaussian fields γ_0 and γ_1 are determined. Figure 4.23 demonstrates the decline of the largest eigenvalues and the estimated energy conservation. The KLEs are truncated at $m_0 = 300$ and $m_1 = 300$ eigenmodes for further steps. They conserve 92.25% and 94.4% of the entire energy.

The PC representations $\rho_{0,c}$ and $\rho_{1,c}$ can now be determined. A comparison between polynomials up to first order and polynomials up to second order for the PCE reveals that the polynomials up to first order are already sufficient: the 2-norm of the difference between the geometrically discretised covariances of the PC representations up to first and second polynomial orders is less than 10^{-4} in its relative value (for each $\rho_{0,c}$ and $\rho_{1,c}$). This insignificant difference is accompanied by a large difference between the numbers of polynomials, namely 301 polynomials up to first order, and $45,451$ polynomials up to second order. Hence, the polynomials up to first order are used for $\rho_{0,c}$ and $\rho_{1,c}$. Figure 4.24 compares the marginal distribution and the covariance function of $\rho_{0,c}$ with the original input F_{r_0} and c_{r_0}; analogous

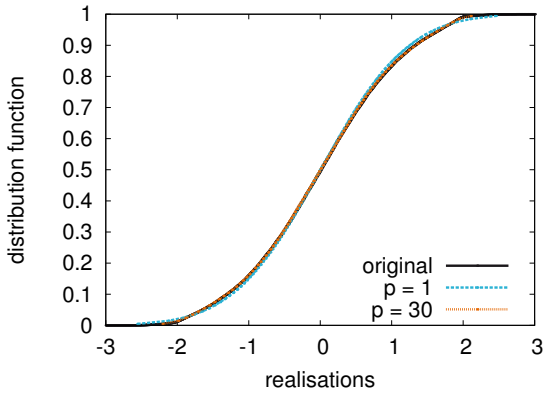
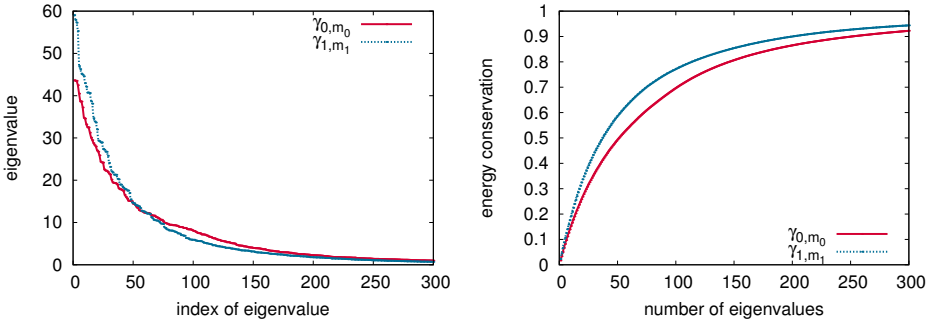


Figure 4.22: *Convergence of $r_{0,p}$ to F_{r_0} by increasing the maximum polynomial order p from 1 to 30. Gauss-Legendre quadrature with $500,000$ sample-points is used for computing $r_{0,p}$.*



(a) (b)

Figure 4.23: *Truncated KL representations γ_{0,m_0} and γ_{1,m_1}: Figure (a) shows the decline of the largest eigenvalues; Figure (b) shows the conserved energy over the number of largest eigenvalues used.*

results are obtained for $\rho_{1,c}$. The quality of $\rho_{0,c}$ or $\rho_{1,c}$ is estimated by comparing the 2-norm of the approximated spatial point variance with the one of the particular input variance of ρ_0 or ρ_1, and by considering the 2-norm of the difference between the geometrically discretised covariance function of the particular approximation and

the particular input covariance function c_{r_0} or c_{r_1}. The estimation identifies 99.34% of variance conservation and 81.99% of covariance conservation for $\rho_{0,c}$ as well as 99.52% of variance conservation and 88.07% of covariance conservation for $\rho_{1,c}$. An essentially higher quality for the covariances is only possible when increasing the numbers m_0 and m_1 of eigenmodes in the KLEs of γ_0 and γ_1; however, the convergence of the conserved energy in Figure 4.23(b) predicts that a quite larger number of eigenmodes may be required.

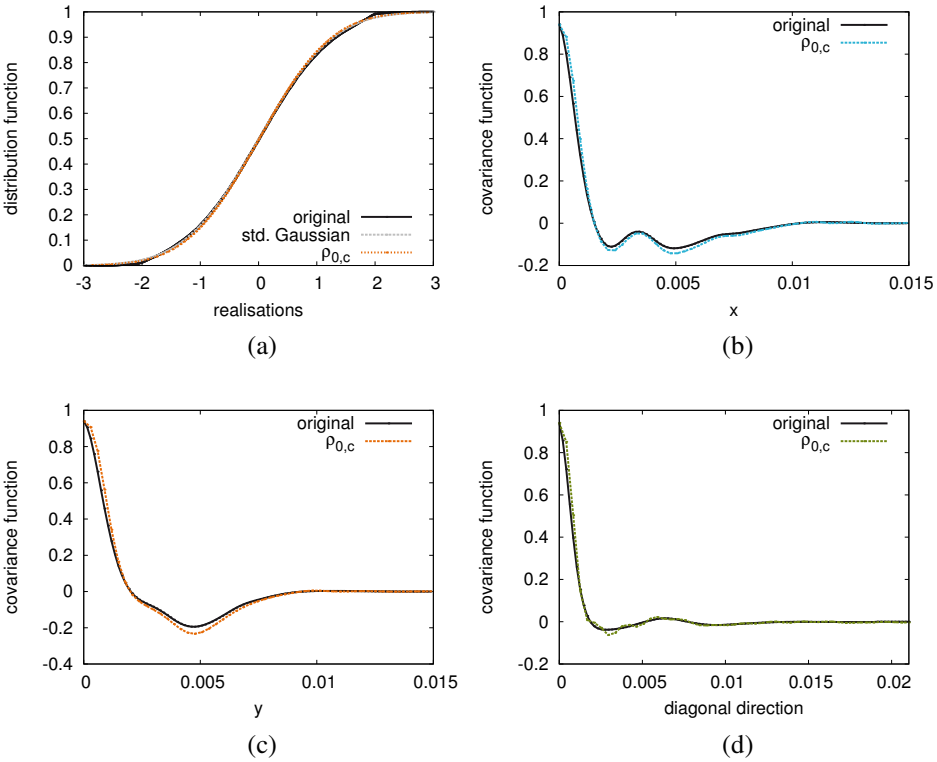


Figure 4.24: *The PC representation $\rho_{0,c}$: Figure (a) compares the marginal distribution of $\rho_{0,c}$ with the original input F_{r_0}; Figures (b), (c), and (d) compare the covariance function of $\rho_{0,c}$ with the original input c_{r_0} along the x- and y-direction and the diagonal direction in the reference plane.*

The truncated PCEs $\rho_{0,c}$ and $\rho_{1,c}$ with stochastic polynomials up to first order are in fact Gaussian fields. This raises the question if the Rosenblatt variables r_0 and r_1

could not have been simply replaced by standard Gaussian random variables. Then, the Rosenblatt fields ρ_0 and ρ_1 could have been directly approximated by truncated KLEs ρ_{0,m_0} and ρ_{1,m_1} without applying the algorithm in [185]. The question will be answered shortly. ρ_{0,m_0} and ρ_{1,m_1} are determined for $m_0 = 300$ and $m_1 = 300$. They conserve 94.87% and 96.4% of the entire energy. (The number of terms is the same in the PC and KL representations, and hence so is their memory requirement.) A quality estimation for ρ_{0,m_0} and ρ_{1,m_1} (analogously to the one for $\rho_{0,c}$ and $\rho_{1,c}$) reveals a slightly better covariance conservation, namely 83.38% for ρ_{0,m_0} and 89.45% for ρ_{1,m_1}. However, the variance conservation is inferior, namely 91.29% for ρ_{0,m_0} and 93.15% for ρ_{1,m_1}. These observations are explained in the following by considering the truncated PC representations $\rho_{0,c}$ and $\rho_{1,c}$ and their process of computation. The input statistics of the Rosenblatt variables indicate that they are only almost Gaussian. This disturbance is present, on the one hand, in the eigenproblems associated with the computations of γ_0 and γ_1 and, on the other hand, in the computation of the PC coefficients by Eq. (4.8). This influence is perceptible in the better variance conservation for $\rho_{0,c}$ and $\rho_{1,c}$. However, the decline of the eigenvalues is slightly slower leading to a lower covariance conservation than for ρ_{0,m_0} and ρ_{1,m_1}.

The Generator Representation for the Interlayer The truncated PC representations $\rho_{0,c}$ and $\rho_{1,c}$ allow to construct the generator $\boldsymbol{g}_{k,c}$ of the stochastic dimension $m := m_0 + m_1 = 600$. $\boldsymbol{g}_{k,c}$ is the first approximated representation obtained for $\boldsymbol{\kappa}_k$. Figure 4.25 shows Young's modulus and the shear modulus corresponding to a sample-point. Estimations $\bar{g}$ and σ_g^2 for the expectation and the variance of $\boldsymbol{g}_{k,c}$ are obtained for each of the twenty-one parameters of the local material tensor by an MC sampling with $2,000,000$ material realisations. These approximations of the stochastic moments are later used to estimate the quality of the truncated PC and KL representations for $\boldsymbol{g}_{k,c}$.

Truncated PC and KL Representations for the Interlayer The truncated PC representation $\boldsymbol{\kappa}_{k,c}$ for $\boldsymbol{\kappa}_k$ is determined by different ways. Projections (see Section 2.4) and a least squares regression approach (see Section 2.5.2) are compared. All these numerical methods sample the generator $\boldsymbol{g}_{k,c}$. It turns out that the regression approach provides the best results.

The projections are based on different integration schemes, namely a basic MC method, a QMC method based on the Halton point sequence, and a Smolyak algorithm based on Gauss-Legendre or Gauss-Hermite quadrature rules. The regression

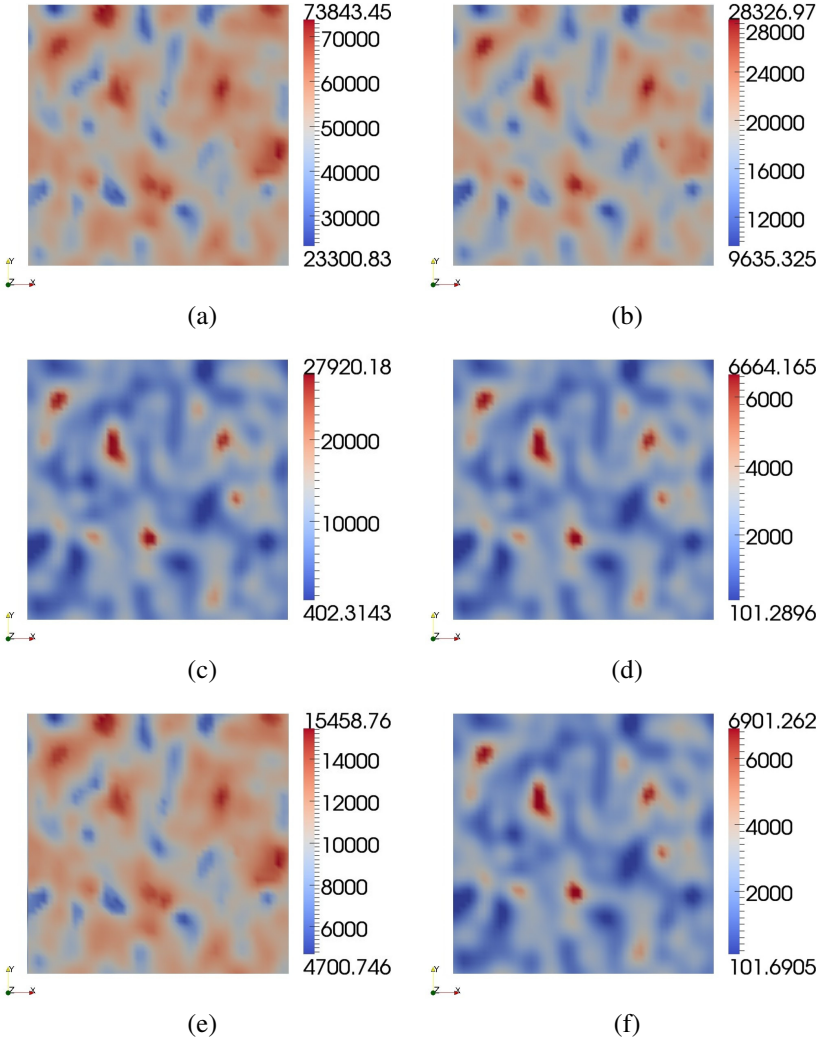


Figure 4.25: *A sample of the k'th interlayer obtained by $\mathbf{g}_{k,c}$: Figures (a), (b), and (c) show Young's modulus in x-, y-, and z-directions; Figures (d), (e), and (f) show the shear modulus in the yz-, xz-, and xy-planes.*

approach uses basic MC samples. The numerical handling is challenging because the stochastic dimension m is comparatively large. Also the finite element discretisation of the geometrical domain and the $N_p := 21$ parameters of the local material tensor lead to $N_X := N_n \times N_p = 2,601 \times 21 = 54,621$ purely geometrical coefficients. The ability of the mentioned numerical methods to determine an accurate $\boldsymbol{\kappa}_{k,c}$ is validated for a truncated PCE with stochastic polynomials up to first order. All of these methods perform more or less well when only the resulting expectations are compared with $\bar{g}$. However, a comparison of the variance with σ_g^2 reveals weaknesses. The projection approach with the basic MC integration based on $1,000$ or $10,000$ samples overestimates the variance by a factor of 27 or 4; the projection approach with the QMC integration based on $1,000$ samples even overestimates the variance by a factor of $17,312$, which is very probably traced back to the comparatively large stochastic dimension. The Smolyak algorithm of the second level identifies $1,201$ samples; it undervalues the variance by a factor of 4 when using Gauss-Legendre quadrature, or by a factor of 1.6 when using Gauss-Hermite quadrature. The regression approach provides the best results. It overestimates the variance only by a factor of 1.03 with $1,000$ MC samples. In other words, it maintains 96.16% of σ_g^2.

Therefore, the regression approach is chosen to determine a truncated PC representation $\boldsymbol{\kappa}_{k,c}$ with stochastic polynomials up to second order. While the PCE with polynomials up to first order is described by 601 polynomials, the PCE up to second polynomial order already identifies $180,901$ polynomials (concerning the set of multi-indices $\mathcal{I}^{mod}$ in Eq. (2.22)). A representation for the PCE with all polynomials up to second order contains $N_X \times 180,901 = 9,880,993,521$ coefficients, which would mean a memory requirement of 73.6 Gigabyte in double precision. However, it is assumed that many polynomials do not have an essential impact on the entire PCE of second order. So the idea is to find the polynomials which have the most impact. The polynomials up to first order have already demonstrated their strong impact. They are chosen as a basic set to measure each of the second order polynomials. For this the basic set is extended by a single second order polynomial. The resulting set of $601 + 1$ polynomials is used in the regression approach with $1,000$ MC samples to obtain an impact measure for the single second order polynomial. The impact measure is simply the 2-norm of the PC coefficients corresponding to the second order polynomial. This process is done for each single second order polynomial separately. The entire process requires the decomposition of $180,300$ dense matrices of dimension $1,000 \times 602$, which were solved N_X-times each. The truncated PC representation $\boldsymbol{\kappa}_{k,c}$ for $\boldsymbol{\kappa}_k$ is finally constructed from the $10,000$ second order polynomials with the highest impact measures, and the polynomials up to first order. The corresponding $10,601$ terms are determined by a regression approach based on $20,000$ MC samples. The computed representation $\boldsymbol{\kappa}_{k,c}$ maintains 99.99% of $\bar{g}$ and 98.97% of σ_g^2. $\boldsymbol{\kappa}_{k,c}$ can be scaled, so that the variance σ_g^2 is completely maintained:

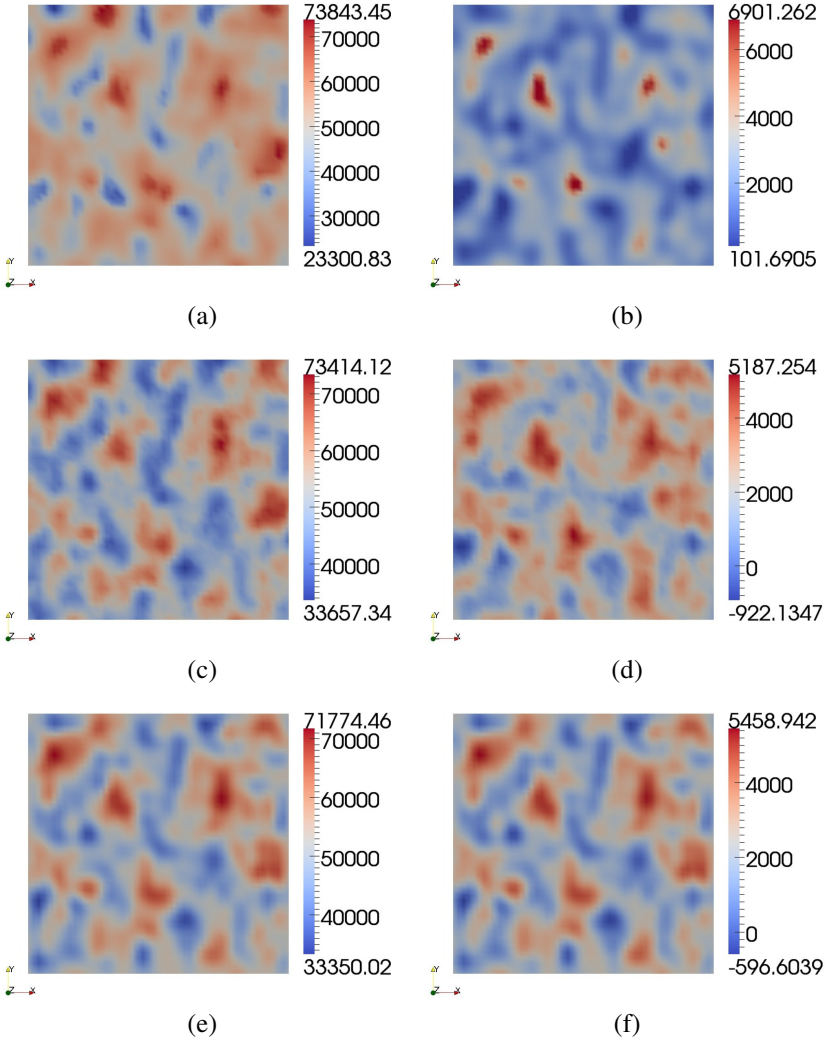


Figure 4.26: *A sample of the k'th interlayer: Young's modulus in x-direction (on the left) and the shear modulus in the xy-plane (on the right) are shown for the generator $\boldsymbol{g}_{k,c}$ in Figures (a) and (b), the PC representation $\boldsymbol{\kappa}_{k,c}$ in Figures (c) and (d), and the KL representation $\boldsymbol{\kappa}_{k,l}$ in Figures (e) and (f); the representations are evaluated at the same sample-point.*

when $\sigma^2_{\boldsymbol{\kappa}_{k,c}}$ is the achieved variance the scale factor is defined as $\frac{\sigma_g^2}{\sigma^2_{\boldsymbol{\kappa}_{k,c}}}$. However, such a scaling is not performed here.

Next, the KL representation $\boldsymbol{\kappa}_{k,l}$ for $\boldsymbol{\kappa}_k$ is determined. The corresponding eigenvalues and eigenvectors were provided by the cooperation partner and result from a projection of the available $170,000$ local material tensor samples onto the eigenfunctions of the Fourier transformation; the truncation criterion was set to 99% of the entire energy, which led to 599 eigenmodes. The uncorrelated random variables inside the KLE are discretised by truncated PCEs. The corresponding PC coefficients are computed by Eq. (2.17) using $\boldsymbol{\kappa}_{k,c}$. The resulting KL representation maintains 99.99% of $\bar{g}$ and 89.89% of σ_g^2. A scaling to completely maintain the variance can be performed in the same manner as explained in the previous passage, but is also not done here.

In contrast to the generator the determined truncated PC and KL representations are not positive-definite anymore (they are still positive-definite in their expectations). This is caused by the truncations. Figure 4.26 shows a sample of Young's modulus in x-direction and the shear modulus in the xy-plane where the latter demonstrates the loss of the positive-definiteness (also negative values appear). The sample of the shear modulus obtained by the KL representation is characterised by the spatial structure which, for instance, occurs for the Young's modulus in the x-direction. This structure is simply the energetically predominant one resolved by the eigenmodes.

When the determined truncated PC or KL representation is used for the material description of a laminated composite structure the application of numerical simulation schemes based on sampling techniques can be queried because a simulation code for deterministic problems may demand positive-definiteness. In contrast, SGM-based schemes may tolerate some infrequently occurring violations of the positive-definiteness. That is demonstrated in Section 4.2.2.4 for a laminated composite structure with a linear constitutive law.

4.2.1.4 Conclusion

An approach to model a fully anisotropic and uncertain carbon-fibre-reinforced composite material for a three-dimensional rigid body is presented in Section 4.2.1. It is based on photographic images of a test specimen which is not externally loaded. It assumes location-independent marginal distributions and homogeneous covariances of the twenty-one parameters which describe the local material tensor. The composite

material is divided into a number of interlayers. An interlayer carries the uncertainty, which is induced by irregular borders between two layers of carbon fibre and the intermediate layer of adhesive, and by the voids within the layer of adhesive. It is described by a non-Gaussian random field. The most accurate representation for that field is provided by a so-called generator which is non-linearly dependent on a set of standard Gaussian random variables. The positive-definiteness of the generator can be assured. PC and KL representations are then derived on the basis of the generator. All these representations are exemplarily determined for a selected interlayer. The corresponding numerical experiments identify a comparatively large dimension of six hundred to describe the interlayer in an acceptable manner. As a consequence of this, the computation of the PC and KL representations is complex. A computationally manageable set of polynomials for the representations is obtained by choosing the polynomials of most significance for a description of the random field from the set of Hermite polynomials up to second order. However, the PC and KL representations determined on the basis of the chosen polynomials are not positive-definite anymore.

4.2.2 A Laminated Composite Structure with a Linear Constitutive Law

A laminated composite structure with a linear constitutive law and a fully anisotropic and partially uncertain material is simulated in this section by different numerical schemes, namely a basic MC method, a least squares regression approach, and the VLR-SR1U scheme. The latter scheme indicates that a relatively high rank is required for an accurate approximation of the solution in the chosen stochastic solution space.

The problem to be solved is introduced in Section 4.2.2.1. Its numerical simulation is presented in the subsequent sections. The results of the basic MC method are discussed in Section 4.2.2.2. The regression approach is applied in Section 4.2.2.3 to obtain an approximation of the solution in the stochastic solution space spanned by polynomials up to first order. The VLR-SR1U scheme searches for a low-rank approximation of the solution in the same stochastic solution space in Section 4.2.2.4. A conclusion is presented in Section 4.2.2.5.

4.2.2.1 Problem Outline

The mentioned laminated composite structure with a linear constitutive law is described by

$$-\nabla_{\boldsymbol{x}} \cdot \boldsymbol{\sigma}(\boldsymbol{x},\omega) = f(\boldsymbol{x}), \qquad \boldsymbol{x} \in X \backslash \partial X \tag{4.10}$$

$$n(\boldsymbol{x}) \cdot \boldsymbol{\sigma}(\boldsymbol{x},\omega) = t(\boldsymbol{x}), \qquad \boldsymbol{x} \in \partial X_N \tag{4.11}$$

$$\boldsymbol{u}(\boldsymbol{x},\omega) = u_D(\boldsymbol{x}), \qquad \boldsymbol{x} \in \partial X_D \tag{4.12}$$

$$\boldsymbol{\sigma}(\boldsymbol{x},\omega) := \boldsymbol{\kappa}(\boldsymbol{x},\omega) : \boldsymbol{\varepsilon}(\boldsymbol{x},\omega) \tag{4.13}$$

$$\boldsymbol{\varepsilon}(\boldsymbol{x},\omega) := \tfrac{1}{2}\left(\nabla_{\boldsymbol{x}}\boldsymbol{u}(\boldsymbol{x},\omega) + (\nabla_{\boldsymbol{x}}\boldsymbol{u}(\boldsymbol{x},\omega))^T\right) \tag{4.14}$$

with $\boldsymbol{x} := (x, y, z) \in X \subset \mathbb{R}^3$ and $\omega \in \Omega$. f and $\boldsymbol{u}$ are the body force and the nodal displacement in the three spatial directions. The local material is described by the fourth-order stiffness tensor $\boldsymbol{\kappa}$, which is expressed by a 6×6 matrix and represented by 21 random variables due to its symmetric structure: $\boldsymbol{\kappa} : \mathbb{R}^3 \times \Omega \to \mathrm{Sym}(\mathbb{R}^{6\times 6})$. $\boldsymbol{\varepsilon}$ and $\boldsymbol{\sigma}$ are the strain and the stress. ":" in Eq. (4.13) indicates the Frobenius product of two tensors of second order. More about the problem setting is specified in the following.

The three-dimensional geometrical domain of the rigid body is defined by $X = [0, l_X] \times [0, w_X] \times [0, h_X]$ with $l_X := l_B$, $w_X := w_B$, and $h_X := 1.094$ millimetres (see definitions for l_B and w_B in Section 4.2.1.1). It is discretised by $23,205$ finite elements. The height h_X results from the layers of the composite material.

The composite material contains fifteen layers, namely eight layers of carbon fibre and seven interlayers. A layer of carbon fibre measures a height of h_{det}, the height for an interlayer is h_{inter} (see definitions in Section 4.2.1.1). In contrast to Section 4.2.1 only the third interlayer is stochastically described. The corresponding stochastic model is the one determined in Section 4.2.1.3. The other interlayers are deterministic and described through the expectations of their actual stochastic models. As a consequence, the composite material is defined through six hundred Gaussian random variables, which simultaneously identify the stochastic dimension of the problem to be solved. The different fibre orientations of the layers are chosen, so that the problem is preferably symmetric. Approximated representations for the local material tensor $\boldsymbol{\kappa}$ are provided through a generator $\boldsymbol{g}_c$ and a truncated KLE $\boldsymbol{\kappa}_l$ (c and l indicate truncations). $\boldsymbol{\kappa}_l$ assures the positive-definiteness only in its expectation, that means single realisations of $\boldsymbol{\kappa}_l$ are not necessarily positive-definite. Furthermore, $\boldsymbol{\kappa}_l$ approximately maintains 89.89% of the variance obtained by $\boldsymbol{g}_c$.

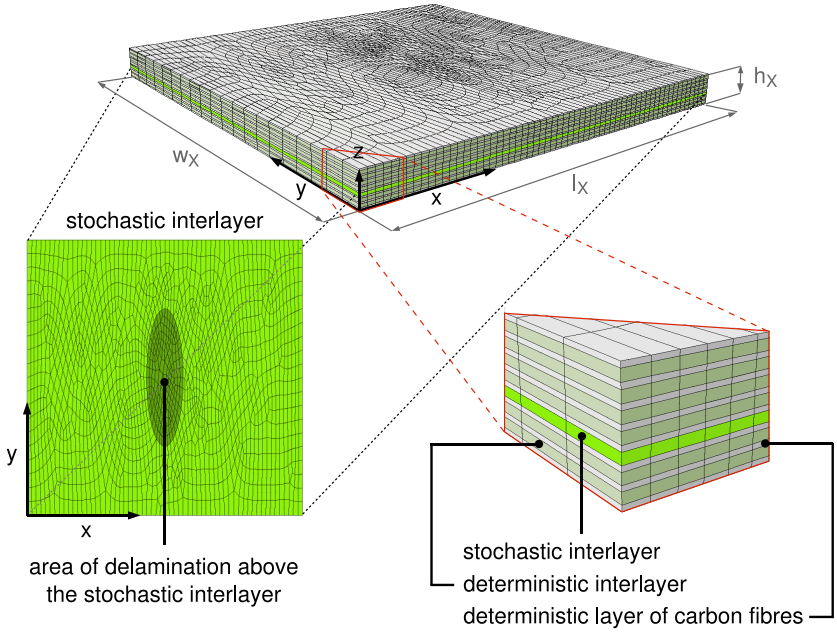


Figure 4.27: *The composite material and the area of the delamination corresponding to the laminated composite structure.*

It is assumed that a delamination between the uncertain interlayer and its overlying layer of carbon fibre occurred during the manufacturing process, that means already before the composite structure is loaded. The delamination is obtained through an elimination of the connectivity in an elliptical area of the finite element mesh. Certainly, that disturbs the symmetry of the problem to be solved. Figure 4.27 illustrates the problem with an emphasis on the composite material and the delamination.

The solution $\boldsymbol{u} := (u_x, u_y, u_z)$ is discretised by $N_X := 78,612$ geometrical DoFs. The essential boundary conditions are deterministic and defined in $\partial X_D := \{(x, y, z) \in X \mid x := 0\}$ as follows:

$$\begin{aligned} \boldsymbol{u}((0,0,0),\omega) &:= (0,0,0) \\ u_z((0,w_X,h_X),\omega) &:= 0 \\ u_x((0,y,z),\omega) &:= 0, \qquad (y,z) \neq (0,0) \wedge (y,z) \neq (w_X,h_X). \end{aligned}$$

The first and second conditions are defined on two single nodes to fix the rigid body

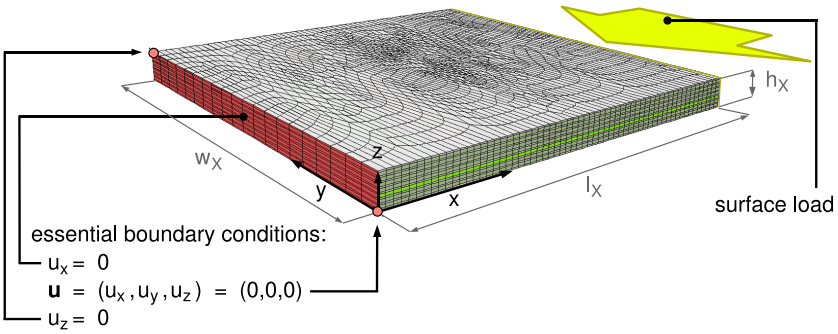


Figure 4.28: *The essential boundary conditions and the external surface load corresponding to the laminated composite structure.*

with regard to a translation and rotation in the geometrical space. The other nodes in ∂X_D are free to move in ∂X_D. The natural boundary conditions are defined at the remaining boundary ∂X_N and set to zero. f establishes an external surface load of -1 on the area $\{(x, y, z) \in X \mid x := l_X\}$. The essential boundary conditions and the external surface load are illustrated in Figure 4.28.

The deterministic simulator component ParaFEP is used in all numerical experiments of this section.

4.2.2.2 A Basic MC Method

A basic MC method is applied to simulate the laminated composite structure. Generator $\boldsymbol{g}_c$ is used for the sampling of the composite material. A total of $116,837$ samples are evaluated.

The expectation of the nodal displacement is presented in Figure 4.29. A torsion of the rigid body can be observed. It is caused by the first and last layers of the composite material, which both are of the same fibre orientation of 45 degrees. The slight distortion in the symmetry of the nodal displacement in z-direction ($6.19 \cdot 10^{-4}$ versus $6.15 \cdot 10^{-4}$) results from the area of the delamination. Figures 4.30(a) and (c) show the expectation and variance of the normal stress σ_{xx} in the stochastic interlayer, that means near the area of the delamination. The expectation indicates larger stresses (in their absolute values) in the area of the delamination. The variance is also higher in that area. The latter demonstrates the need to model the uncertainty of a

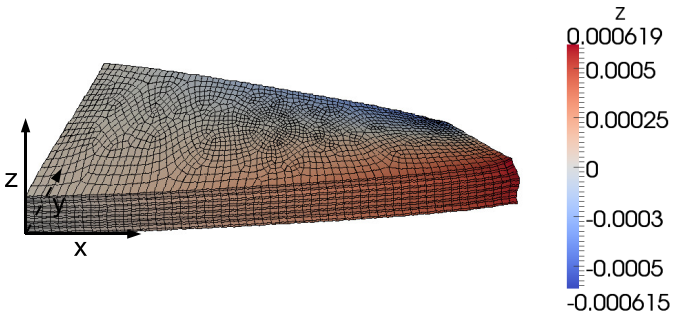


Figure 4.29: *The expectation of the nodal displacement obtained by a basic MC method with* $116,837$ *samples: the mesh displays the expectation of the nodal displacement in all spatial directions and is scaled by a factor of* $3,000$*, while the colour plot is not scaled and belongs only to the expectation of the nodal displacement in* z*-direction.*

composite material, because the larger variance influences a progression of delamination. The changes near the boundary in Figure 4.30(a) are traced back to the finite element mesh, see Figure 4.30(b).

The 2-norms of the spatial point expectation and the spatial point variance of the nodal displacement are computed to get reference solutions for comparisons with further numerical schemes. (The 2-norm of the spatial point expectation means the 2-norm of the expectations at the discrete FE nodes; analogously for the 2-norm of the spatial point variance.) The central limit theorem provides error estimations for these references. It predicts a probability larger than 0.9999 that the relative error for the expectation is smaller than 10^{-4}, and the one for the variance is smaller than 10^{-2}. The reference solutions are also used to display the convergence of the basic MC method itself, see Figure 4.31.

4.2.2.3 A Least Squares Regression Approach

A least squares regression approach (see Section 2.5.2) is applied to construct a surrogate model for the probabilistic model of the laminated composite structure, which is based on $N_S := 601$ Hermite polynomials up to first order. As a consequence, $N_S \times N_X = 47,245,812$ coefficients have to be computed. For the construction of the surrogate model the regression approach uses basic MC evaluations of the probabilistic model on the basis of $\boldsymbol{g}_c$; different numbers of evaluations are tried.

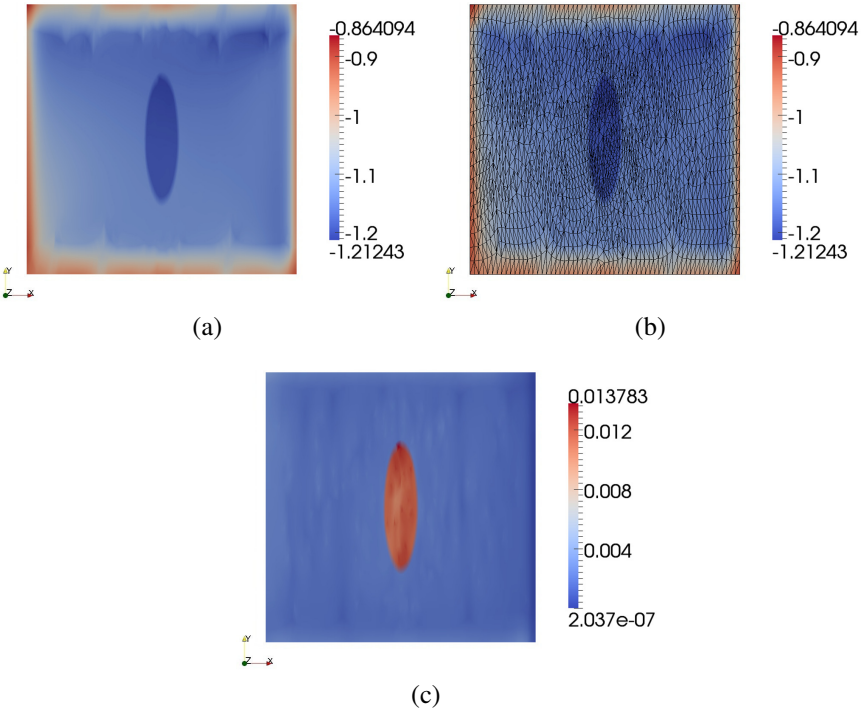


Figure 4.30: *The expectation and variance of the stress σ_{xx} in the stochastic interlayer obtained by a basic MC method with $116,837$ samples: the expectation is shown in Figures (a) and (b) without and with displaying the finite element mesh; the corresponding variance is shown in Figure (c).*

The spatial point expectation and the spatial point variance are analytically computed from the surrogate model. For the 2-norm of the spatial point expectation the applied regression approach with $1,000$ evaluations of the probabilistic model already reaches the accuracy of the MC reference solution which was obtained by $116,837$ evaluations (see Section 4.2.2.2). In contrast, the regression approach with $10,000$ evaluations resolves approximately only 77.7% of the MC reference solution for the 2-norm of the spatial point variance. However, a strongly higher accuracy for that quantity does not seem to be reachable in the chosen stochastic solution space with polynomials up to first order. That can be concluded from the following circumstances. On the one hand, both the regression approach and the basic MC method use $\boldsymbol{g}_c$ to generate samples of the material. The regression approach, on the other hand,

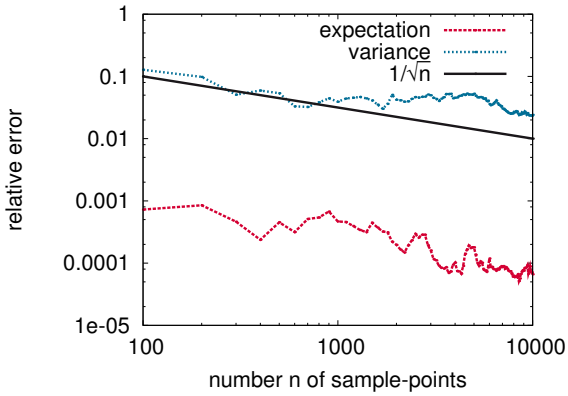


Figure 4.31: *Convergence of a basic MC method with regard to the relative error of the* 2*-norms of the spatial point expectation and the spatial point variance of the nodal displacement; the reference solution is obtained by* $116,837$ *samples.*

converges to a 2-norm of the spatial point variance which differs already slightly in the first digit from the one computed by the basic MC method, see Figure 4.32. In that figure the 2-norms of the spatial point variances are directly shown instead of showing relative errors.

The 2-norm of the spatial point variance obtained by the regression approach with $10,000$ evaluations of the probabilistic model is used as a reference solution in the successive section, because there the VLR-SR1U scheme is applied to compute a low-rank approximation of the solution in the same stochastic solution space spanned by Hermite polynomials up to first order.

4.2.2.4 The VLR-SR1U Scheme

The VLR-SR1U scheme is applied to search for an accurate low-rank approximation for the solution of the laminated composite structure. The stochastic solution space is chosen as in Section 4.2.2.3. The composite material is described by the KL representation $\boldsymbol{\kappa}_l$ which maintains 89.89% of the variance provided by $\boldsymbol{g}_c$, see Section 4.2.1.3.

Successive rank-one updates are performed by the basic VLR-SR1U scheme up to rank 450, which is almost three quarters of the full rank. The convergence for the

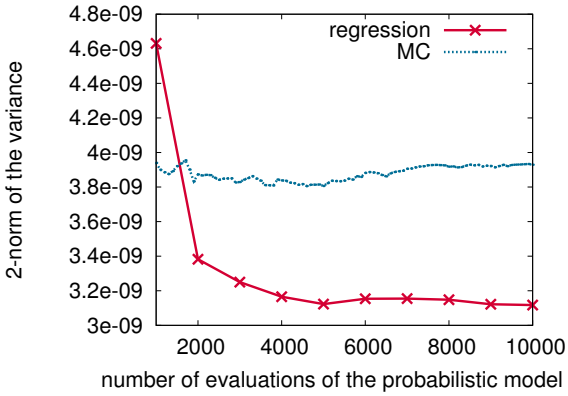


Figure 4.32: *Convergence of the regression approach for the* 2*-norm of the spatial point variance (absolute values are plotted instead of errors) in comparison to the corresponding convergence of the basic MC method.*

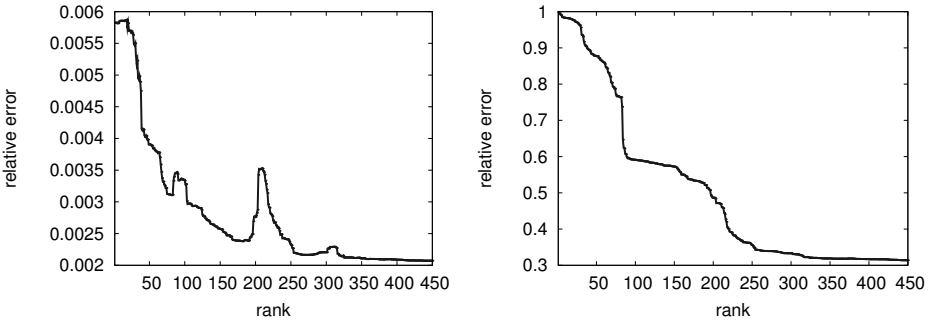


Figure 4.33: *Convergence of the basic VLR-SR1U scheme with regard to the relative error of the* 2*-norms of the spatial point expectation in Figure (a) and the spatial point variance in Figure (b); the reference solution is obtained by a regression approach with* $10,000$ *evaluations of the probabilistic model.*

2-norms of the spatial point expectation and the spatial point variance are shown in Figure 4.33, where the reference solution to compute relative errors was obtained by the regression approach with $10,000$ evaluations of the probabilistic model presented in Section 4.2.2.3. Because the stochastic solution spaces used for the regression ap-

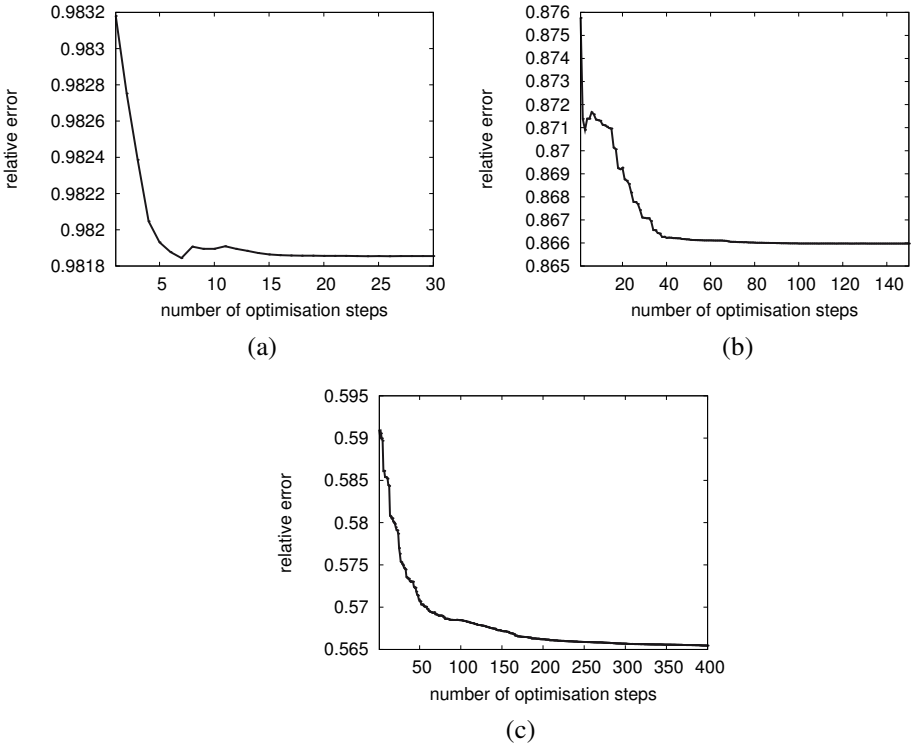


Figure 4.34: *Convergence of the VLR-OPT scheme with regard to the relative error of the* 2*-norm of the spatial point variance: approximations of rank* 10*,* 50*, and* 100*, which are determined by the basic VLR-SR1U scheme, are optimised in Figures (a), (b), and (c); the reference solution is obtained by a regression approach with* $10,000$ *evaluations of the probabilistic model.*

proach and the VLR-SR1U approach are the same there are two circumstances which decrease the accuracy of the low-rank approximation of the solution in comparison to the regression approach. These are the application of the KL representation for describing the composite material and the suboptimality of the basic VLR-SR1U scheme. It can be observed in Figure 4.33 that 99.79% of the 2-norm of the spatial point expectation and only 68.6% of the 2-norm of the spatial point variance are reached at the final rank (or 53.4% of the 2-norm of the spatial point variance with respect to the reference solution obtained by the basic MC method, see Section 4.2.2.2). Certainly, the KL representation for the composite material causes an accuracy loss

for the solution (which may also be of significant impact) but, nevertheless, the slow convergence of the basic VLR-SR1U scheme remains: the first digit of the 2-norm of the spatial point variance is not fixed until rank 220, that is approximately one-third of the full rank.

However, the computed low-rank approximations are not optimal. The VLR-OPT scheme is applied to optimise some low-rank approximations of different ranks. The corresponding convergence is presented in Figure 4.34 for the 2-norm of the spatial point variance. The optimisation of the approximations of rank 10, 50, and 100 only leads to accuracy gains of approximately 0.13%, 0.98%, and 2.55%. In consideration of these results it can be concluded that an accurate approximation for the solution of the laminated composite structure in the chosen stochastic solution space requires a relatively high rank.

4.2.2.5 Conclusion

The stochastic model determined in Section 4.2.1.3 is used in this section to describe the third interlayer of a fully anisotropic material of a laminated composite structure in a three-dimensional geometrical space with a linear constitutive law. The remaining layers of the composite material are assumed to be deterministic. A generator and a truncated KLE are available representations for the composite material. Furthermore, an area of a delamination between the third interlayer and the overlying layer of carbon fibre is introduced. The simulation of the composite structure is obtained by a basic MC method, a least squares regression approach, and the VLR-SR1U scheme.

The MC method identifies a larger variance of stresses in the area of the delamination, which influences a progression of delamination. This emphasises the need to model the uncertain characteristics of a composite material. Additionally, the MC method provides reference solutions for further numerical methods. The regression approach is used to project the solution onto a stochastic solution space which is spanned by Hermite polynomials up to first order. The projected solution maintains approximately 77.7% of the variance which was computed by the MC method. The accuracy loss is a consequence of the low polynomial order chosen for the stochastic solution space. The basic VLR-SR1U scheme is applied to search for a low-rank approximation of the solution in the same stochastic solution space. Some low-rank approximations are optimised by the VLR-OPT scheme. However, approximately one-third of the full rank is already required so that the first digit of the 2-norm of the spatial point variance is fixed. Consequently, a relatively high rank is needed to accurately approximate the solution of the laminated composite structure in the chosen stochastic solution space with Hermite polynomials up to first order.

4.3 Aircraft Design with Uncertain Parameters

This section studies the design of a future low-noise, economically competitive civil aircraft for short runways and with improved performance characteristics. A high-lift system is applied which combines a blown flap concept with fuel-efficient turboprop engines. The aircraft design is strongly influenced by two factors: first, high lift is required while take-off and landing to cope with a short runway. Second, the blown flap concept has an impact on the engine design.

An uncertainty description for parameters of the aircraft model, which are essentially associated with the mentioned factors, is introduced. The outcome is a probabilistic model for the aircraft design. The simulation code PrADO is used as a black box to simulate deterministic samples of the probabilistic model. PrADO operates on a deterministic parameterised model description for the aircraft. The uncertainty propagation is quantified through a basic MC method. An access to many distributed PrADO instances is established through the component implementation coPrADO (see Section 3.4.4.3). However, the simulation of a deterministic aircraft sample is computationally expensive. Therefore, a PC representation is computed to obtain a surrogate model. A least squares regression approach (see Section 2.5.2) is applied to compute the PC coefficients. On the one hand the surrogate model can be quickly evaluated, on the other hand its construction requires far less evaluations of the probabilistic model than the MC method to provide statistics of the same accuracy.

General phases of an aircraft design are outlined in Section 4.3.1. The parameterised aircraft model and its iterative process in PrADO are discussed in Section 4.3.2. Section 4.3.3 describes the reference model of the aircraft, which is the basis for the uncertainty quantification. The uncertain parameters are expressed through independent stochastic noise parameters, which are introduced in Section 4.3.4. The MC method is applied in Section 4.3.5, and a surrogate model for the computationally expensive probabilistic model is determined. A conclusion is presented in Section 4.3.6.

This section refers to the Collaborative Research Centre (CRC) 880 “Fundamentals of High Lift for Future Civil Aircraft” funded by the “DFG - Deutsche Forschungsgemeinschaft” (see `http://www.dfg.de`). The reference model for the aircraft, the list of uncertain parameters, and an adapted version of PrADO — which enables the percentage setting of the uncertain parameters — were provided by the cooperation partner. Ranges for the uncertain parameters were jointly decided.

4.3.1 Phases in Aircraft Design

Before an aircraft is manufactured a design process needs to be carried out. The process may be divided into three phases [177, 173], namely the *conceptual design*, the *preliminary design*, and the *detail design*, in which the level of detail more and more increases. The conceptual design draws up main attributes like the performance and fuel consumption, and the size and weight of the fuselage, the wing, and the tail. The preliminary design considers these attributes in finer detail; for instance the fuselage obtains initial values for its length and radius, its skin and stringers are introduced, and thicknesses are initialised. The detail phase prepares the manufacturing process by determining the remaining individual characteristics like ribs and spars, and by designing an assembly plan. In each phase the particularly resulting aircraft configuration needs to meet the requirements of the customer. The uncertainty quantification carried out in this thesis is positioned in the preliminary design.

4.3.2 PrADO's Parameterised Aircraft Model

The program structure of PrADO is commented on in Section 3.4.4.3. This section focuses PrADO's parameterised model of an aircraft [82, 81]. The parameterised model is initially an idea of the aircraft to be designed and is iterated to obtain a substantiated model. An iteration step determines, for instance, the current aircraft geometry, computes the aerodynamic characteristics, sizes the propulsion, simulates the entire flight spectrum of the aircraft to estimate the fuel consumption, performs structural load tests, computes the masses of the component parts, and calculates the direct operation costs (DOC) for the life-cycle of the aircraft.

The parameterised model consists of independent and dependent design parameters. The set of independent design parameters is the input given by the user; these parameters are constants marking the requirements of the aircraft. Independent design parameters are, for instance, transport-oriented ones like the range, the maximum runway lengths, the number of passengers, the payload mass, and the cruise speed, or they are geometry-oriented ones to describe the arrangement and properties of the aircraft components. Dependent design parameters may depend on independent or dependent ones. The latter dependency formulates a mutual relation between dependent design parameters. Dependent design parameters are for instance the empty mass, aerodynamic coefficients, and the demand for thrust and fuel associated to the entire flight spectrum of the aircraft.

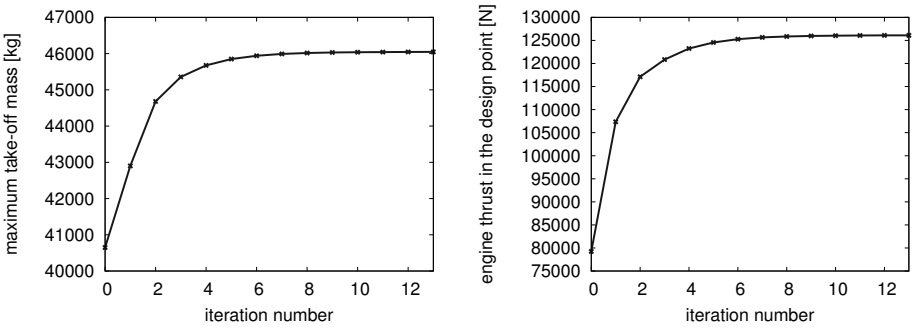


Figure 4.35: *Convergent sizing during the iterative process of PrADO demonstrated at different dependent design parameters: Figure (a) shows the maximum take-off mass; Figure (b) shows the engine thrust in the design point (see Table 4.7)*

PrADO starts with an initialisation of the parameterised model and iterates until the dependent design parameters reach a specified precision. The iterative process indicates a sizing. When, for instance, the mass and the volume (and consequently the surface) increase locally (for example of the wing) the corresponding global quantities increase. Furthermore, this implies more required thrust and, accordingly, more fuel. The associated mass of fuel rises, therefore the take-off mass increases. In this manner the dependent design parameters are mutually building up from iteration to iteration, the so-called "snowball effect". However, a technically feasible aircraft configuration leads to a convergent behaviour of the mutual parameters. Usually 5–10 iterations are required in PrADO to obtain the substantiated model of the aircraft. Figure 4.35 demonstrates the convergent iterative sizing. The sizing is traced back to the *square-cube law* [44, 100, 209]. The square-cube law is exemplified in its simplest case by a square cube: a square cube with mass m and volume V is loaded by a force F (it is assumed that F is proportional to the mass); when the size of the square cube is varied by the length scale factor s (in each spatial direction), then V, m, and accordingly F vary with the scale factor s^3, while the cross section area — resisting the load — varies with factor s^2, and accordingly the stress varies linearly with factor s. The resulting variation of the stress has a consequence: the square-cube law limits the sizing of a loaded physical object (or subject) if a tolerable stress for the object (or subject) is limited. That is of course the case for aircrafts.

The iterative process of PrADO leads to a single analysis of an aircraft design. It is noted in this context and in addition to Section 3.4.4.3 that PrADO also offers mathematical methods for other modes: a parameter variation of the independent design parameters can be performed to analyse sensitivities, an optimisation of the independent design parameters can be performed concerning a specific objective function.

4.3.3 The Reference Model of the Aircraft

The requirements for the civil aircraft in the CRC 880 are listed in Table 4.7. The runway lengths are more precisely the take-off field length and the landing field length. A deterministic reference model of the aircraft satisfying the requirements was provided by the cooperation partner. It is specified by A_{ref}, which is the converged parameterised aircraft model determined by the iterative process $\mathcal{P}$ of PrADO and an initial guess A_0 of the aircraft:

$$A_{ref} := \mathcal{P}(A_0).$$

Figure 4.36 shows the reference model.

4.3.4 The Master Case and Stochastic Noise Parameters

Uncertainty is introduced to the reference model — specified by A_{ref} — through independent stochastic noise parameters, which act onto design parameters (see Section 4.3.2) of the model. As the noise parameters perturb the reference model, A_{ref} can be denoted as the master case. Table 4.8 summarises the key features of the master case.

A perturbed master case needs to pass the iterative process of PrADO to result in a converged aircraft design. That means the noise parameters establish the iterative process of a stochastic parameterised aircraft model

$$A(\omega) = \mathcal{P}_s(A_{ref}, \omega)$$

with $\omega \in \Omega$. $\mathcal{P}_s$ is the iterative process of the preliminary design, which is adapted to enable the integration of noise parameters. The input model A_{ref} marks the master case to be perturbed. Fourteen independent uniformly distributed noise parameters

Design requirements for the aircraft in the CRC 880	
First year of operation	2025
Certification according to FAR 25	
Comfort standard of Airbus A320	
Flight with maximum payload (= design case)	
Range	$2,000\ km$
Payload (100 passengers with baggage and 2200 kg additional freight)	$12,000\ kg$
Flight with 100 passengers and without additional freight	
Range	$\geq 2,800\ km$
Flight with maximum fuel	
Range is determined by the available tank volume in the wing box	
Cruise conditions (optimised with respect to minimum DOC)	
Initial cruise altitude	$10,600\ m$
Cruise Mach number	$0.74\ m$
FAR 25 take-off and landing field lengths (SL/ISA)	$\leq 800\ m$

Table 4.7: *Design requirements for the civil aircraft in the CRC 880 (this information is kindly provided by Wolfgang Heinze, Institute of Aircraft Design and Lightweight Structures, TU Braunschweig). FAR 25, SL, and ISA are abbreviations for Federal Aviation Regulations — part 25 [61], sea level, and international standard atmosphere. Direct operating costs (DOC) are costs which are directly allocated to the aircraft.*

θ_i ($i \in \{1, \ldots, 14\}$) are defined, so that the essential factors for a high lift and the corresponding influence on the engine design can be stochastically quantified. They are specified in Table 4.9. The perturbation interval is measured in percentage, for example θ_2 perturbs the mass of the fuselage between -5% and 5%.

Noise parameters θ_1 and θ_2 affect dependent design parameters. θ_3, θ_4, and θ_5 act onto design parameters associated with the aerodynamic computation involved in $\mathcal{P}_s$. These design parameters are actually dependent ones, but the aerodynamic computation is here deactivated for the uncertainty quantification. As a consequence these

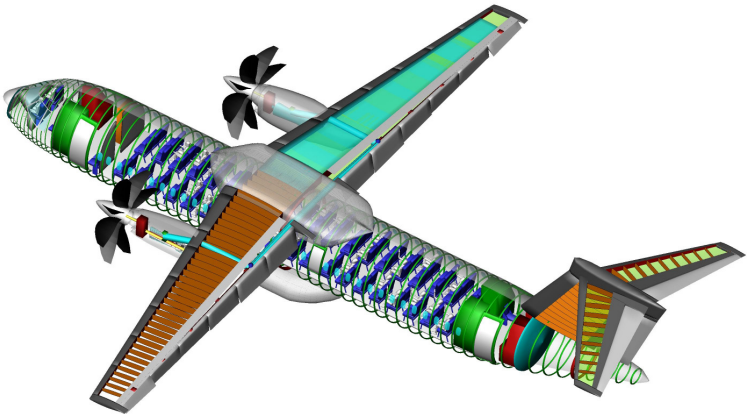


Figure 4.36: *The reference design of the aircraft in the CRC 880 (this picture is kindly provided by Wolfgang Heinze, Institute of Aircraft Design and Lightweight Structures, TU Braunschweig).*

design parameters behave like independent ones, meaning they are perturbed only once (at the beginning) in $\mathcal{P}_s$. The aerodynamic computation is deactivated to reduce the runtime for simulating an aircraft sample under the assumption that the influence of the uncertainty is low concerning aerodynamics. The noise parameter θ_6 perturbs the mass of the propulsion system, which is actually an independent design parameter. However, here it is a dependent design parameter because of a possible thrust adaption which may be required to satisfy the runway lengths. θ_7 to θ_{12}, θ_{13}, and θ_{14} act onto independent design parameters.

The noise can reflect two different kinds of uncertainties, namely in a data assumption or in a method of calculation. Practically speaking, the result of the method is perturbed in the latter case. Thus the perturbation can be performed repetitively, that means once in each iteration of $\mathcal{P}_s$. A sample of such a parameter certainly leads either to an under- or overestimation during the entire iterative process $\mathcal{P}_s$.

Due to the described handling the following holds: when $\mathcal{P}_s$ is executed for a sample

Master case: key features		
description	value	unit of measurement
Maximum engine shaft power	$8,396.296$	kW
Maximum bleed air extraction for		
. the aircraft's systems	0.880	kg/s
. the blown flap system	14.444	kg/s
. Total	15.324	kg/s
Specific fuel consumption during cruising	$4.593 \cdot 10^{-2}$	$kg/N/h$
Maximum lift coefficient, landing configuration	4.5	–
Lift-to-drag ratio during cruising	14.491	–
Operating empty mass:		
. Mass of the wing	$3,498.819$	kg
. Mass of the fuselage	$5,407.543$	kg
. Mass of the propulsion system	$4,541.743$	kg
. Mass of the blown flap system	204.564	kg
. Total	$23,927.793$	kg
Total fuel mass (design case)	$4,721.758$	kg
Maximum take-off mass	$40,649.551$	kg
Maximum landing mass	$38,855.217$	kg
Range with maximum fuel	$8,012.977$	km
FAR 25 take-off field length	779.432	m
FAR 25 landing field length	758.622	m
Approach speed in landing configuration	50.355	$\frac{m}{s}$
Direct Operating Costs (DOC, design case)	$7.951 \cdot 10^{-2}$	USD/skm

Table 4.8: *Key features of the master case (CRC 880 reference model from June 2011): the DOC is measured in US Dollar per revenue seat-kilometre (USD/skm).*

Stochastic noise parameters			
No.	perturbed design parameter	type	[min,max] noise in [%]
Structure:			
1	Mass of the wing	D,M	$[-5, 5]$
2	Mass of the fuselage	D,M	$[-5, 5]$
Aerodynamics:			
3	Lift coefficient	M	$[-5, 5]$
4	Lift-induced drag coefficient	M	$[-5, 5]$
5	Zero-lift drag coefficient (mainly viscous drag)	M	$[-5, 5]$
Propulsion system:			
6	Mass of the propulsion system	M	$[-5, 5]$
7	Polytropic efficiency of the HPC	D	$[-3, 3]$
8	Polytropic efficiency of the HPT	D	$[-3, 3]$
9	Polytropic efficiency of the LPT	D	$[-3, 3]$
10	Combustion efficiency	D	$[-3, 0]$
11	Total pressure ratio of the HPC	D	$[-3, 3]$
12	Maximum turbine inlet temperature	D	$[-3, 3]$
Blown flap system:			
13	Mass of the blown flap system	M	$[-50, 100]$
14	Mass flow rate for the blown flap system	D	$[-5, 5]$

Table 4.9: *Specifications for the fourteen independent stochastic noise parameters: minimum and maximum noises are presented for each design parameter to be perturbed; types "D" and "M" indicate if an uncertainty is induced by the data or the corresponding method of calculation. Abbreviations HPC, HPT, and LPT mean high pressure compressor, high pressure turbine, and low pressure turbine.*

of the noise parameters the independent design parameters (or independently occurrent design parameters) exhibit exactly the input perturbance at the end of the process, while the dependent design parameters exhibit a stronger decrease or increase because of the snowball effect inside $\mathcal{P}_s$ (see Section 4.3.2).

4.3.5 Numerical Experiments

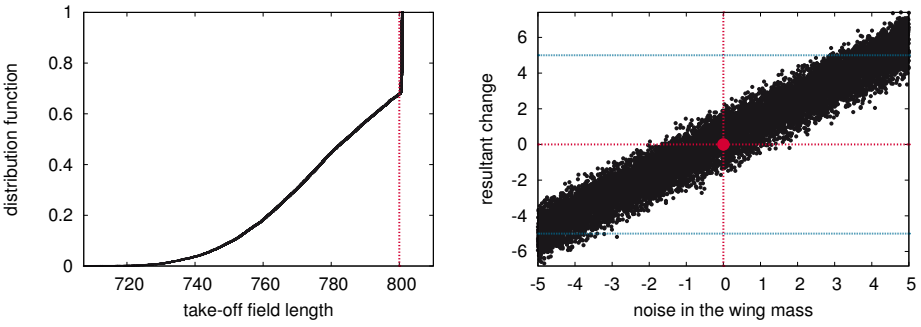
The influence of the stochastic noise parameters on the aircraft design is measured here by means of some design parameters, which are listed as output parameters in Table 4.10. The first six parameters reflect the influence on the technology, the last parameter considers the influence on the economic efficiency. Their uncertainty is quantified through a basic MC method and discussed in the first paragraph. An efficient surrogate model for the probabilistic model is constructed in the second paragraph.

Output parameters	
No.	output parameter
1	Maximum engine shaft power
2	Operating empty mass (total)
3	Total fuel mass (design case)
4	Maximum take-off mass
5	Lift-to-drag ratio during cruising
6	FAR 25 landing field length
7	DOC (design case)

Table 4.10: *Output parameters: the corresponding units of measurement can be found in Table 4.8.*

Basic MC Method A basic MC method evaluated $12,191$ samples, from which four were not converging. (A non-convergence of samples is caused by the fact that the reference model of the aircraft is close to some constraints.) As a consequence, the following results are obtained from the $12,187$ converging simulations. The MC simulation used three computing machines with eight processors working in parallel, thus all in all twenty-four processors were used. Each processor had its own instance of coPrADO. The expectation and the standard deviation of the runtime of one simulation were $4.203\ h$ and $1.106\ h$.

First, a special configuration of $\mathcal{P}_s$ is commented, which causes an observable effect in some quantified data. Then, another (independent) effect is exemplified. $\mathcal{P}_s$ is triggered so that the requirement of the maximum runway lengths (see Table 4.7) is



(a) (b)

Figure 4.37: *Effects which occure within the uncertainty quantification of the aircraft: Figure (a) shows the distribution function concerning the take-off field length, where the red line marks the thrust adaption; Figure (b) shows the correlation between the initial noise on the mass of the wing in percent (induced through the first stochastic noise parameter, see Table 4.9) and its resulting change in percent (obtained in the iterative process of PrADO), where the red dot indicates the master case and the blue lines mark the boundaries of the noise.*

preferably established for the deterministic aircraft samples in the numerical experiments: $\mathcal{P}_s$ adapts the thrust appropriately when otherwise the requirement would not be met. This thrust adaption is activated in approximately 32% of the samples, and in these cases the maximum take-off field length is exactly met. This has an effect to stochastic quantities, which concentrate on the take-off, see for instance Figure 4.37(a). Furthermore, when an aircraft parameter is initially perturbed, then the snowball effect in the iterative process of PrADO usually causes a larger perturbation of the same parameter. That effect is demonstrated in Figure 4.37(b).

Next, statistics for the output parameters are addressed, a convergence behaviour of the MC method is considered in the subsequent paragraph. Stochastic moments are presented in Tables 4.11 and 4.12. The first table also contains error estimates obtained through the central limit theorem; the errors of the expectations and the standard deviations vary with a factor of approximately 10^{-2}. Figures 4.38(a), (b), and (c) present the distribution functions for the maximum engine shaft power (the first output parameter), the total mass of fuel (the third output parameter), and the landing field length (the sixth output parameter). It can be seen that

Statistics and estimated errors					
No.	master case	expectation	error (probability)	standard deviation	error
1	$8,396.296$	$8,537.557$	10^{-3} (0.991)	357.151	10^{-1}
2	$23,927.793$	$24,066.826$	10^{-3} (0.999)	362.864	10^{-1}
3	$4,721.758$	$4,837.241$	10^{-3} (0.978)	232.404	10^{-1}
4	$40,649.551$	$40,904.067$	10^{-3} (0.999)	540.475	10^{-1}
5	14.491	14.519	10^{-3} (0.999)	$3.613 \cdot 10^{-1}$	10^{-1}
6	758.622	761.166	10^{-3} (0.999)	10.150	10^{-1}
7	$7.951 \cdot 10^{-2}$	$8.016 \cdot 10^{-2}$	10^{-3} (0.999)	$1.077 \cdot 10^{-3}$	10^{-1}

Table 4.11: *Statistics and estimated errors: expectations and standard deviations of the output parameters are compared to the corresponding parameters of the master case. The output parameters indexed in the first column are listed in Table 4.10; the corresponding units of measurement can be seen in Table 4.8. The error of each stochastic moment is estimated by the central limit theorem. The probabilities that the errors of the expectations are correct are given in brackets; the probabilities corresponding to the errors of the standard deviations are all at least* 0.999*. Probabilities are always rounded down.*

the first output parameter is at the most 7.22% larger than the master case for 90% of the realisations. Analogously, 8.9% holds for the third output parameter, and even smaller percentages hold for the other output parameters, for instance 2.52% for the direct operation costs (DOC, the seventh output parameter). The demanded landing field length is not met for 0.04% of the samples with a maximum deviation of only 5.06 metres. Some correlations are demonstrated in Figure 4.39; Figure (a) displays the correlation between the lift coefficient and the maximum engine shaft power. It is obvious that a reduction of the lift coefficient leads to a stronger increase of the demanded power. That is indirectly a consequence of the convention to establish the requirement of the maximum take-off field length. This convention influences also the DOC and is demonstrated in Figure 4.39(b) through the correlation between the lift coefficient and the DOC: when a higher lift coefficient is usable a less powerful (and accordingly less expensive) engine is required. In total it can be concluded that the reference design of the aircraft proves to be very robust.

Statistics and estimated errors		
No.	deviation between master case and expectation in [%]	std. deviation in [%]
1	1.682	4.183
2	0.581	1.508
3	2.446	4.804
4	0.626	1.321
5	0.188	2.488
6	0.335	1.334
7	0.816	1.344

Table 4.12: *Percentage deviations: the deviation between the expectation of the output parameters and the appropriate parameters of the master case are presented in the second column; the percentage standard deviation is presented in the third column. The output parameters indexed in the first column are listed in Table 4.10, corresponding units of measurement can be seen in Table 4.8.*

A Surrogate Model The evaluation of the probabilistic model $A(\omega)$ is computationally expensive. A surrogate model $\tilde{A}(\omega)$ is constructed from a set of samples of $A(\omega)$. When $\tilde{A}(\omega)$ is constructed it can be quickly evaluated to obtain statistics. However, the statistics which are considered here are analytically computed from $\tilde{A}(\omega)$ and compared to the ones computed by the basic MC method. It is demonstrated that $\tilde{A}(\omega)$ provides as accurate statistics as the basic MC method, but requires less evaluations of $A(\omega)$.

$\tilde{A}(\omega)$ is simply a truncated PC expansion. The stochastic basis polynomials are Legendre polynomials defined on uniformly distributed random variables. The corresponding coefficients are computed by the least squares regression approach — explained in Section 2.5.2 — on a set of evaluations of $A(\omega)$ obtained by the basic MC sampling. The surrogate model is constructed for different numbers of evaluations and different maximum polynomial orders. The influence of these construction parameters on the quality of the surrogate model is demonstrated by a comparison of statistics obtained from the surrogate model and the basic MC method. As already mentioned, the statistics from the surrogate model are analytically computed.

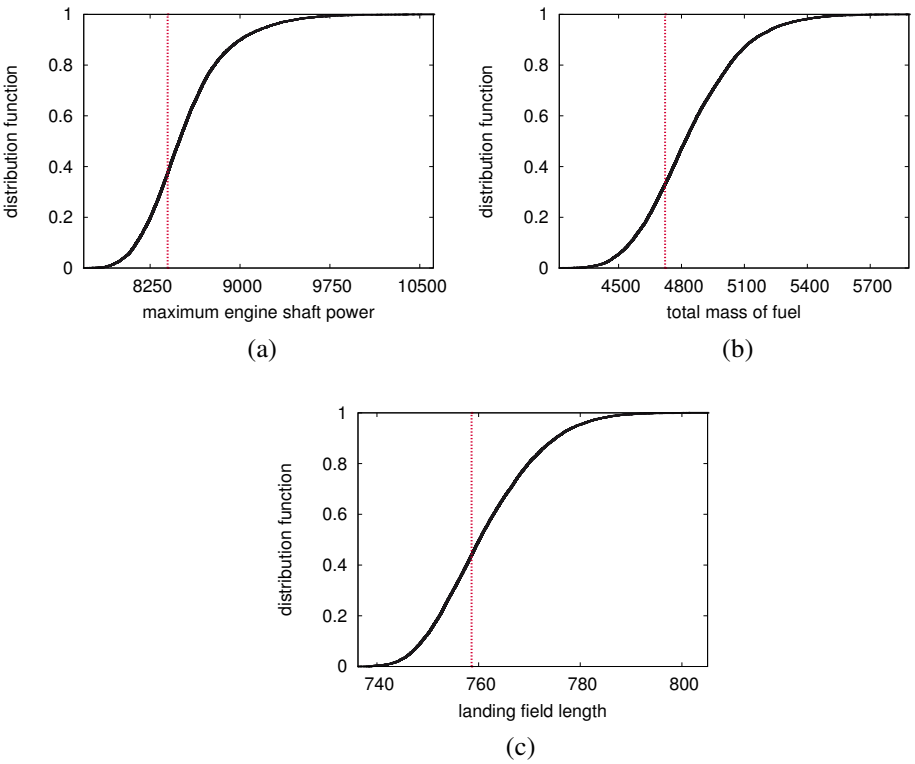
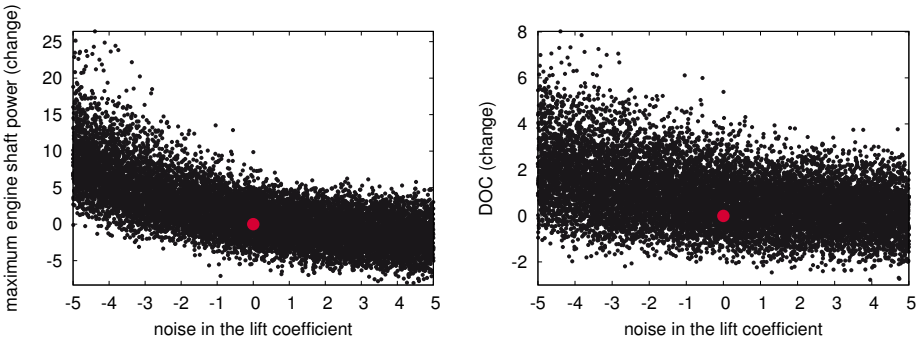


Figure 4.38: *Distribution functions for some output parameters: Figures (a), (b), and (c) show the distribution functions for the maximum engine shaft power, the total mass of fuel (in the design case), and the landing field length; the references of the master case are marked in red.*

References are provided by the basic MC method with $12,187$ evaluations of $A(\omega)$, presented in the previous section. Figure 4.40 exemplarily shows the convergence of the expectation and the variance of the first and the third output parameters (see Table 4.10); these output parameters exhibit the largest standard deviations in relation to their expectations (see Table 4.12). The first number of model evaluations, which is used to construct the surrogate model, is the number of stochastic basis polynomials incremented by one so that the system to be solved in the regression approach is over-determined. The figures display that a few more model evaluations than the



(a) (b)

Figure 4.39: *Correlations for some parameters measured in their percentage change, the references of the master case are marked in red: Figure (a) shows the correlation between the lift coefficient and the maximum engine shaft power; Figure (b) shows the correlation between the lift coefficient and the direct operation costs (DOC) in the design case.*

minimum number are required to reach the accuracy of the MC references. Figure 4.40(b) shows that a maximum order of two for the stochastic basis polynomials is necessary to reach an accurate variance for the first output parameter. All in all, the regression approach with polynomials up to second order achieves the accuracy of the considered expectations and variances with less evaluations of $A(\omega)$ than the basic MC method requires.

4.3.6 Conclusion

A probabilistic model for an efficient future civil aircraft for short runways is derived from a deterministic reference model and simulated in the previous sections. Fourteen independent uniformly distributed noise parameters are introduced to describe the uncertainty of the aircraft model, see Section 4.3.4. They act onto the relevant factors, which influence the aircraft design. Statistics of the probabilistic models are determined through a basic MC method, see Section 4.3.5. The deterministic simulator component coPrADO is used in many distributed instances and interfaces with simulation code PrADO. The statistics reveal the uncertain aircraft model to be robust towards the considered noises.

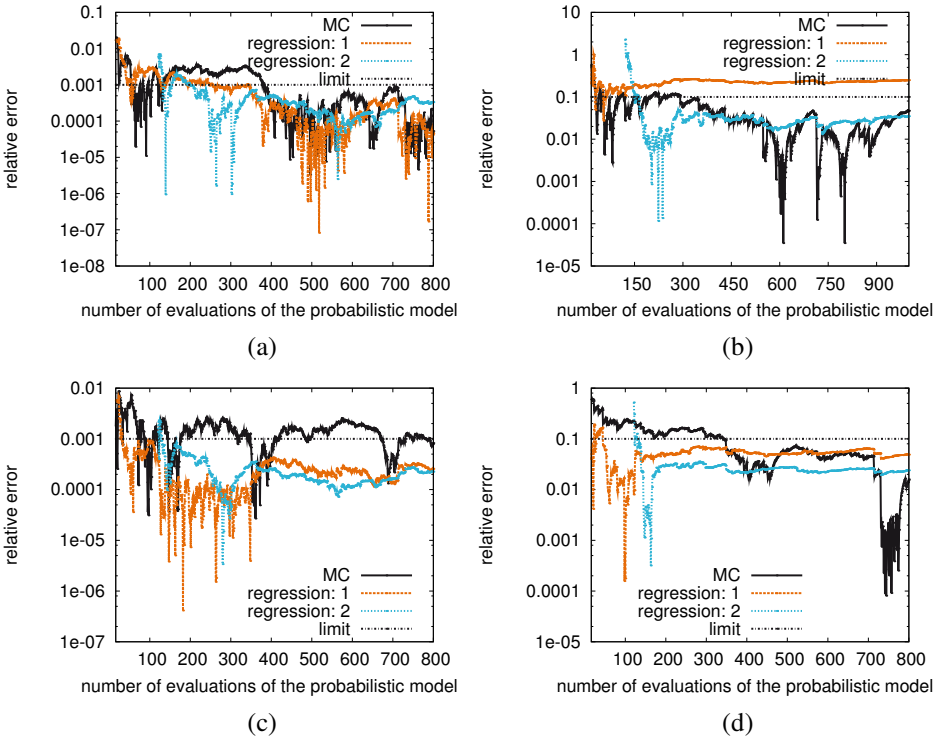


Figure 4.40: *Convergence of the surrogate model in comparison with a basic MC method with respect to the number of evaluations of the probabilistic model: Figures (a) and (b) show the expectation and the variance corresponding to the maximum engine shaft power (the first output parameter); Figures (c) and (d) show the same for the total mass of fuel in the design case (the third output parameter). The surrogate model contains either stochastic basis polynomials up to first ("1") or second ("2") order. The confidence limits of the MC references obtained by the central limit theorem are plotted as well, see Table 4.11. Below these limits statements about the convergence cannot be made.*

The sampling of the probabilistic model is computationally expensive. A truncated PC representation for the probabilistic model is computed by a least squares regression approach, and is based on Legendre polynomials. It can be used as a surrogate model, for instance, to compute statistics. It requires less evaluations of the probabilistic model for its construction to provide statistics of the same quality like the basic MC method, see Section 4.3.5.

4.4 Conclusion

The chapter of the numerical experiments focuses on three different application-specific subjects, namely a groundwater flow, a laminated composite material (and a corresponding structure), and an aircraft design. For all cases uncertain parameters are stochastically modelled and corresponding probabilistic problems are simulated. The simulation is performed through the frameworks of the distributed generic component-based software architecture presented in Chapter 3.

A stationary groundwater flow problem with an uncertain hydraulic conductivity is considered in Section 4.1. This section emphasises the convergence behaviour of the VLR-SR1U scheme in its basic form and its extended form through the VLR-OPT and RBSSE schemes. The VLR-SR1U scheme is compared to direct integration schemes like MC and QMC methods as well as the Smolyak algorithm. It demonstrates its potential to be an efficient scheme and exhibits far better convergence than the (Q)MC methods.

A fully anisotropic and uncertain model for a laminated composite material is presented in Section 4.2, which is based on photographic images of a three-dimensional test specimen. It requires a comparatively large stochastic dimension to provide an acceptable description of the input statistics. A corresponding probabilistic problem of a laminated composite structure with a linear constitutive law is simulated through different numerical schemes. Statistics and references are obtained through a basic MC method. A least squares regression approach projects the solution on a stochastic solution space spanned by polynomials up to first order, which already keeps a larger percentage of the references. The VLR-SR1U scheme is applied to search for a low-rank approximation in the same stochastic solution space. However, an accurate approximation requires a relatively high rank.

Uncertain parameters are introduced to a deterministic model of a future civil aircraft in Section 4.3 to investigate its robustness. The resulting probabilistic model is simulated through a basic MC method to gain statistics and references. The statistics confirm the robustness of the aircraft design. However, the evaluation of the probabilistic model is quite expensive. Therefore, a surrogate model is constructed through a least squares regression approach. It provides statistics in the same quality as the basic MC method, but requires far less evaluations of the probabilistic model. The deterministic simulator component — used in the simulation — is also introduced in this thesis.

Chapter 5

Conclusion and Outlook

This thesis provides contributions for several challenging issues in an uncertainty quantification of probabilistic models. These are a meaningful formulation of a model, efficient numerical schemes for its simulation, and a sustainable software implementation. The contributions are separately considered in the next paragraphs. Each paragraph contains the corresponding outlook in the last passage.

A Successive Rank-One Update with a Global Optimisation and an Adaptive Solution Space Construction Some numerical schemes are contributed, namely the basic VLR-SR1U scheme, the VLR-OPT scheme, and the RBSSE scheme. They are associated with the system which results from an SGM discretisation of a stochastic elliptic partial differential equation. The basic VLR-SR1U scheme approximates a low-rank representation for the solution of the system through successive rank-one updates in the assumption that the expectation of the total potential energy is minimised. It is a kind of a greedy approach in the sense that a rank-one update tries to extract the most remaining energy. A rank-one update is obtained by an alternating algorithm, which is similar to the known alternating least squares algorithm. The successive updates enable an error-controlled rank increase until a required accuracy is reached. A resulting approximation of rank r is, however, not the minimiser of the mentioned energy compared to all approximations of rank r. This suboptimal approximation is addressed by the VLR-OPT scheme. It optimises a given approximation of rank r through rank-one update matrices so that the approximation becomes the minimiser without increasing the rank. The scheme is derived from the algorithm of the basic VLR-SR1U scheme. The basic VLR-SR1U scheme is extended by an application of the VLR-OPT scheme to optimise approximations either during the process of rank updates or not until a final rank is reached. The RBSSE scheme rates new stochastic basis polynomials — which are still not used in

the current representation of the solution — for their ability to describe the solution. It determines an extended residual for the current solution which already involves new stochastic polynomials. This residual is transferred to the unit of measurement of the solution, so that relevant new stochastic polynomials can be identified. The VLR-SR1U scheme is extended by an application of the RBSSE scheme to adaptively construct the solution space. The schemes are presented in Section 2.7.2. The numerical efficiency of the schemes is demonstrated in Section 4.1.4. The basic VLR-SR1U scheme and its extension by the VLR-OPT scheme are compared to direct integration schemes. They clearly beat (Q)MC methods, but do not reach the best configuration of the Smolyak algorithm.

The adaptive construction of the solution space currently follows a straight forward strategy, for instance 10% more stochastic basis polynomials are used in each rank update. More sophisticated strategies need to be developed which determine if it is more reasonable to perform a rank-one update or to extend the solution space.

Distributed Generic Component-Based Software Architecture A software architecture is contributed for the simulation of probabilistic models and presented in Chapter 3. This architecture is based on distributed generic software components, so that it gains from the capacity of a parametric polymorphism in distributed software systems. In this manner, the strengths of a generic programming and a component-based distribution of processes are combined. It is stated here that the approach enables one to keep the essence of generic types also at runtime without any need of recompilation. Corresponding software frameworks are developed for a direct integration ((Q)MC methods, the Smolyak algorithm), an integration-based projection, several collocation schemes (a sparse stochastic Lagrange interpolation and a least squares regression approach), and SGM-based schemes. The VLR-SR1U scheme, the VLR-OPT, and RBSSE schemes are implemented in this context as well.

A separation of concerns leads to a number of components, which are reused in different frameworks and may even be reused beyond an uncertainty quantification. Components are interchangeable at runtime and may be instantiated in a distributed system many times. An essential component is the deterministic simulator component, which is reused in each framework. Its interface reflects the general conceptions for a simulation or construction of a deterministic model. Its implementation identifies individual conceptions, and it usually interfaces with a (third-party) simulation code. Bindings to actual third-party simulation codes are available for a preliminary aircraft design and for problems of groundwater flow or structural mechanics. The

deterministic simulator component may also be used beyond an uncertainty quantification, for instance in an optimisation.

Currently, the locations of involved software components need to be set up by the user. This could be extended to an automatic detection of the locations.

An Uncertain Composite Material A model for an uncertain, anisotropic composite material in a three-dimensional geometrical domain is contributed and presented in Section 4.2. It is constructed from marginal distribution and covariance functions, which were extracted from photographic images and provided by a cooperation partner. The composite material is virtually divided in deterministic and stochastic layers, where each layer corresponds to its own model. The model for a concrete stochastic layer is numerically constructed in a computationally expensive process. A comparatively large stochastic dimension of six hundred is required for an acceptable description of the input statistics. Different representations for the model are obtained, namely a simple generator, a truncated PCE, and a truncated KLE.

A laminated composite structure with a linear constitutive law and a partially uncertain material (one stochastic layer) is simulated through a basic MC method to gain statistics. A least squares regression approach is applied to obtain an approximation in a stochastic solution space spanned by polynomials up to first order. This space already resolves a large amount of the variance. The basic VLR-SR1U scheme and the VLR-OPT scheme are applied to search for a low-rank approximation in the same space, however an accurate approximation requires a relatively high rank in that solution space.

On the one hand the results of the simulation reveal the need for an uncertainty quantification in the field of composite materials. On the other hand more than one stochastic layer would be required to describe a composite material more meaningfully. Accordingly, the resulting stochastic dimension would be the sum of the stochastic dimensions for all stochastic layers and, as mentioned before, the stochastic dimension of a single stochastic layer is already comparatively large. As a consequence, methods for a dimension reduction may be considered for further steps.

An Uncertain Aircraft Design An uncertainty quantification for a future civil aircraft is contributed and presented in Section 4.3. For this purpose a probabilistic model is formulated for a deterministic reference design of the aircraft. It is simulated by a basic MC method to obtain statistics. The evaluation of the probabilistic

model is quite expensive. A surrogate model is represented by a truncated PCE and determined by a straight forward least squares regression approach — without detecting a sparse PCE. Its construction requires far less aircraft samples than the MC method to compute statistics of the same accuracy.

However, a more accurate approximation of the solution can be described in stochastic solution spaces of higher polynomial order. As a sampling of the probabilistic model is expensive, a straight forward least squares regression approach limits the applicable polynomial orders. A strategy for a detection of a sparse PCE is the next step.

List of Figures

List of Tables

Bibliography

[1] LangPop.com — programming language popularity. http://www.langpop.com/, 2011.

[2] The computer language benchmarks game. http://shootout.alioth.debian.org/, 2012. Copyright Brent Fulgham.

[3] Swagato Acharjee and Nicholas Zabaras. A non-intrusive stochastic Galerkin approach for modeling uncertainty propagation in deformation processes. *Computers & Structures*, 85(5–6):244–254, 2007. http://dx.doi.org/10.1016/j.compstruc.2006.10.004.

[4] Malcolm Adams and Victor Guillemin. *Measure Theory and Probability*. Birkhäuser, Boston, 1996.

[5] Robert J. Adler. *The Geometry of Random Fields*. John Wiley & Sons, Chichester, 1981.

[6] Robert J. Adler and Jonathan E. Taylor. *Random Fields and Geometry*. Springer Monographs in Mathematics. Springer Science + Business Media LLC, New York, 2007. http://dx.doi.org/10.1007/978-0-387-48116-6.

[7] Amine Ammar, Francisco Chinesta, and Antonio Falcó. On the convergence of a greedy rank-one update algorithm for a class of linear systems. *Archives of Computational Methods in Engineering*, 17(4):473–486, 2010. http://dx.doi.org/10.1007/s11831-010-9048-z.

[8] Edward Anderson, Zhaojun Bai, Christian Bischof, Susan Blackford, James Demmel, Jack Dongarra, Jeremy Du Croz, Anne Greenbaum, Sven Hammarling, Alan McKenney, and Danny Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, 3rd edition, 1999.

[9] Rob Armstrong, Gary Kumfert, Lois Curfman McInnes, Steven Parker, Ben Allan, Matt Sottile, Thomas Epperly, and Tamara Dahlgren. The CCA component model for high-performance scientific computing. *Concurrency and Computation: Practice and Experience*, 18(2):215–229, 2006. http://dx.doi.org/10.1002/cpe.911.

[10] Holger Arndt, Markus Bundschus, and Andreas Naegele. Towards a next-generation matrix library for Java. In *33rd Annual IEEE International Computer Software and Applications Conference*, pages 460–467, Washington, 2009. IEEE Computer Society. http://dx.doi.org/10.1109/COMPSAC.2009.67.

[11] Ivo Babuška, Fabio Nobile, and Raúl Tempone. A stochastic collocation method for elliptic partial differential equations with random input data. *SIAM Journal on Numerical Analysis*, 45(3):1005–1034, 2007. http://dx.doi.org/10.1137/050645142.

[12] Ivo Babuška, Raúl Tempone, and Georgios E. Zouraris. Galerkin finite element approximations of stochastic elliptic partial differential equations. *SIAM Journal on Numerical Analysis*, 42(2):800–825, 2005. http://dx.doi.org/10.1137/S0036142902418680.

[13] Jonas Ballani and Lars Grasedyck. A projection method to solve linear systems in tensor format. *Numerical Linear Algebra with Applications*, 20(1):27–43, 2012. http://dx.doi.org/10.1002/nla.1818.

[14] Robert Balzer and Neil Goldman. Principles of good software specification and their implications for specification languages. In *Proceedings of the May 4-7, 1981, National Computer Conference*, AFIPS '81, pages 393–400, New York, 1981. Association for Computing Machinery (ACM). http://dx.doi.org/10.1145/1500412.1500468.

[15] Helmut Balzert. *Lehrbuch der Softwaretechnik: Entwurf, Implementierung, Installation und Betrieb*. Spektrum Akademischer Verlag, Heidelberg, 3rd edition, 2011.

[16] Richard H. Barnard and David R. Philpott. *Aircraft flight: A Description of the Physical Principles of Aircraft Flight*. Pearson Education Limited, Harlow, 4th edition, 2010.

[17] Len Bass, Paul Clemens, and Rick Kazman. *Software Architecture in Prac-*

tice. The SEI Series in Software Engineering. Addison-Wesley, Boston, 2nd edition, 2003.

[18] Sarah C. Baxter and Lori L. Graham. Characterization of random composites using moving-window technique. *Journal of Engineering Mechanics*, 126(4):389–397, 2000. http://dx.doi.org/10.1061/(ASCE)0733-9399(2000)126:4(389).

[19] Stefan Behnel, Robert Bradshaw, Craig Citro, Lisandro Dalcin, Dag Sverre Seljebotn, and Kurt Smith. Cython: The best of both worlds. *Computing in Science and Engineering*, 13(2):31–39, 2011. http://dx.doi.org/10.1109/MCSE.2010.118.

[20] Marc Berveiller, Bruno Sudret, and Maurice Lemaire. Stochastic finite element: A non intrusive approach by regression. *European Journal of Computational Mechanics*, 15(1–3):81–92, 2006.

[21] Gregory Beylkin and Martin J. Mohlenkamp. Numerical operator calculus in higher dimensions. *Proceedings of the National Academy of Sciences of the United States of America (PNAS)*, 99(16):10246–10251, 2002. http://dx.doi.org/10.1073/pnas.112329799.

[22] Gregory Beylkin and Martin J. Mohlenkamp. Algorithms for numerical analysis in high dimensions. *SIAM Journal on Scientific Computing*, 26(6):2133–2159, 2006. http://dx.doi.org/10.1137/040604959.

[23] Marcel Bieri and Christoph Schwab. Sparse high order FEM for elliptic SPDEs. *Computer Methods in Applied Mechanics and Engineering*, 198(13–14):1149–1170, 2009. http://dx.doi.org/10.1016/j.cma.2008.08.019.

[24] Géraud Blatman and Bruno Sudret. Sparse polynomial chaos expansions and adaptive stochastic finite elements using a regression approach. *Comptes Rendus Mécanique*, 336(6):518–523, 2008. http://dx.doi.org/10.1016/j.crme.2008.02.013.

[25] Géraud Blatman and Bruno Sudret. An adaptive algorithm to build up sparse polynomial chaos expansions for stochastic finite element analysis. *Probabilistic Engineering Mechanics*, 25(2):183–197, 2010. http://dx.doi.org/10.1016/j.probengmech.2009.10.003.

[26] Géraud Blatman and Bruno Sudret. Adaptive sparse polynomial chaos expansion based on least angle regression. *Journal of Computational Physics*, 230(6):2345–2367, 2011. http://dx.doi.org/10.1016/j.jcp.2010.12.021.

[27] Joshua Bloch. *Effective Java*. The Java Series. Addison-Wesley, Boston, 2nd edition, 2008.

[28] Helmut Brass. *Quadraturverfahren*. Vandenhoeck & Ruprecht, Göttingen, 1977.

[29] Paul Bratley and Bennett L. Fox. Algorithm 659: Implementing Sobol's quasirandom sequence generator. *ACM Transactions on Mathematical Software (TOMS)*, 14(1):88–100, 1988. http://dx.doi.org/10.1145/42288.214372.

[30] William L. Briggs. *A Multigrid Tutorial*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, 1994.

[31] Matteo Broggi and Gerhart I. Schuëller. Efficient modeling of imperfections for buckling analysis of composite cylindrical shells. *Engineering Structures*, 33(5):1796–1806, 2011. http://dx.doi.org/10.1016/j.engstruct.2011.02.019.

[32] Barry W. Brown and James Lovato. *RANDLIB.C — library of C routines for random number generation*. Department of Biomathematics, University of Texas, Houston, 1997. Version 1.3.

[33] Boris Bügling. The CTL protocol and its Java implementation. Student research project, Institut für Wissenschaftliches Rechnen, Technische Universität Braunschweig, Braunschweig, 2006. http://www.wire.tu-bs.de/forschung/projekte/ctl/files/ctl_spec.pdf.

[34] Boris Bügling and Martin Krosche. Coupling the CTL and MATLAB. Informatikbericht 2007–03, Institut für Wissenschaftliches Rechnen, Technische Universität Braunschweig, Braunschweig, 2007. http://www.digibib.tu-bs.de/?docid=00021292.

[35] Hans-Joachim Bungartz and Stefan Dirnstorfer. Higher order quadrature on sparse grids. In *Lecture Notes in Computer Science*, volume 3039, pages 394–401, Berlin, 2004. Computational Science - ICCS 2004, Kraków, Poland,

Springer. http://dx.doi.org/10.1007/978-3-540-25944-2_52.

[36] Hans-Joachim Bungartz and Michael Griebel. Sparse grids. In *Acta Numerica*, volume 13, pages 147–269. Cambridge University Press, Cambridge, 2004.

[37] Russel E. Caflisch. Monte Carlo and quasi-Monte Carlo methods. In *Acta Numerica*, volume 7, pages 1–49. Cambridge University Press, Cambridge, 1998.

[38] Wesley J. Cantwell and John Morton. The impact resistance of composite materials — a review. *Composites*, 22(5):347–362, 1991. http://dx.doi.org/10.1016/0010-4361(91)90549-V.

[39] Nian-Zhong Chen and Carlos Guedes Soares. Spectral stochastic finite element analysis for laminated composite plates. *Computer Methods in Applied Mechanics and Engineering*, 197(51–52):4830–4839, 2008. http://dx.doi.org/10.1016/j.cma.2008.07.003.

[40] Yannis Chicha, Michael Lloyd, Cosmin E. Oancea, and Stephen M. Watt. Parametric polymorphism for computer algebra software components. In *6th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, SYNASC '04, pages 119–130, Timisoara, 2004.

[41] Francisco Chinesta, Amine Ammar, and Elías Cueto. Recent advances and new challenges in the use of the proper generalized decomposition for solving multidimensional models. *Archives of Computational Methods in Engineering*, 17(4):327–350, 2010. http://dx.doi.org/10.1007/s11831-010-9049-y.

[42] Francisco Chinesta, Pierre Ladeveze, and Elías Cueto. A short review on model order reduction based on proper generalized decomposition. *Archives of Computational Methods in Engineering*, 18(4):395–404, 2011. http://dx.doi.org/10.1007/s11831-011-9064-7.

[43] Doo Bo Chung, Miguel A. Gutiérrez, and René de Borst. Object-oriented stochastic finite element analysis of fibre metal laminates. *Computer Methods in Applied Mechanics and Engineering*, 194(12–16):1427–1446, 2005. http://dx.doi.org/10.1016/j.cma.2004.03.021.

[44] Frank A. Cleveland. Size effects in conventional aircraft design. *Journal of Aircraft*, 7(6):483–512, 1970.

[45] Krzysztof Czarnecki and Ulrich W. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, Boston, 2000.

[46] Lisandro D. Dalcin, Rodrigo R. Paz, Pablo A. Kler, and Alejandro Cosimo. Parallel distributed computing using Python. *Advances in Water Resources*, 34(9):1124–1139, 2011. http://dx.doi.org/10.1016/j.advwatres.2011.04.013.

[47] Brian P. Danowsky, Jeffery R. Chrstos, and David H. Klyde. Evaluation of aeroelastic uncertainty analysis methods. *Journal of Aircraft*, 47(4):1266–1273, 2010. http://dx.doi.org/10.2514/1.47118.

[48] Christopher Greenough David Worth and Lee Shawn Chin. A survey of C and C++ software tools for computational science. Technical report, Science and Technology Facilities Council, Didcot, 2009. http://www.softeng.rl.ac.uk/media/uploads/publications/2010/03/c-c_tools_report.pdf.

[49] Mark Day, Robert Gruber, Barbara Liskov, and Andrew C. Myers. Subtypes vs. where clauses: Constraining parametric polymorphism. In *Proceedings of the Tenth Annual Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, volume 30 of *OOPSLA '95*, pages 156–168, New York, 1995. Association for Computing Machinery (ACM). http://dx.doi.org/10.1145/217838.217852.

[50] Daniel A. DeLaurentis, Dimitri N. Mavris, and Daniel P. Schrage. System synthesis in preliminary aircraft design using statistical methods. In *20th Congress of the International Council of the Aeronautical Sciences (ICAS)*, pages 1–13, Sorrento, 1996. Georgia Institute of Technology. http://hdl.handle.net/1853/6282.

[51] Jacobo Díaz and Santiago Hernández. Uncertainty quantification and robust design of aircraft components under thermal loads. *Aerospace Science and Technology*, 14(8):527–534, 2010. http://dx.doi.org/10.1016/j.ast.2010.04.004.

[52] Josef Dick, Gunther Leobacher, and Friedrich Pillichshammer. Randomized Smolyak algorithms based on digital sequences for multivariate integration.

IMA Journal of Numerical Analysis, 27(4):655–674, 2007. http://dx.doi.org/10.1093/imanum/drm002.

[53] John Doherty. *PEST: Model-independent parameter estimation — user manual*, 5th edition, 2005/2010. http://www.pesthomepage.org/.

[54] Alireza Doostan, Roger G. Ghanem, and John Red-Horse. Stochastic model reduction for chaos representations. *Computer Methods in Applied Mechanics and Engineering*, 196(37–40):3951–3966, 2007. http://dx.doi.org/10.1016/j.cma.2006.10.047.

[55] Alireza Doostan and Gianluca Iaccarino. A least-squares approximation of partial differential equations with high-dimensional random inputs. *Journal of Computational Physics*, 228(12):4332–4345, 2009. http://dx.doi.org/10.1016/j.jcp.2009.03.006.

[56] Alireza Doostan and Houman Owhadi. A non-adapted sparse approximation of PDEs with stochastic inputs. *Journal of Computational Physics*, 230(8):3015–3034, 2011. http://dx.doi.org/10.1016/j.jcp.2011.01.002.

[57] Michael Eiermann, Oliver G. Ernst, and Elisabeth Ullmann. Computational aspects of the stochastic finite element method. *Computing and Visualization in Science*, 10(1):3–15, 2007. http://dx.doi.org/10.1007/s00791-006-0047-4.

[58] Michael S. Eldred, Brian M. Adams, David M. Gay, Laura P. Swiler, Karen Haskell, William J. Bohnhoff, John P. Eddy, William E. Hart, Jean-Paul Watson, Patricia D. Hough, and Tammy G. Kolda. *DAKOTA, A multilevel parallel object-oriented framework for design optimization, parameter estimation, uncertainty quantification, and sensitivity analysis — developers manual*. Sandia National Laboratories, Livermore, 2006/2008. http://dakota.sandia.gov/index.html.

[59] Burçin Eröcal and William Stein. The Sage project: Unifying free mathematical software to create a viable alternative to Magma, Maple, Mathematica and MATLAB. In *Proceedings of the Third International Congress Conference on Mathematical Software*, ICMS'10, pages 12–27, Berlin, 2010. Springer. http://dl.acm.org/citation.cfm?id=1888390.1888395.

[60] Mike Espig, Wolfgang Hackbusch, Thorsten Rohwedder, and Reinhold

Schneider. Variational calculus with sums of elementary tensors of fixed rank. *Numerische Mathematik*, 122(3):469–488, 2012. http://dx.doi.org/10.1007/s00211-012-0464-x.

[61] Federal Aviation Administration (FAA). Federal Aviation Regulations — part 25 (FAR 25). http://www.faa.gov.

[62] Jasmine Foo and George E. Karniadakis. Multi-element probabilistic collocation method in high dimensions. *Journal of Computational Physics*, 229(5):1536–1557, 2010. http://dx.doi.org/10.1016/j.jcp.2009.10.043.

[63] Jasmine Foo, Xiaoliang Wan, and George E. Karniadakis. The multi-element probabilistic collocation method (ME-PCM): Error analysis and applications. *Journal of Computational Physics*, 227(22):9572–9595, 2008. http://dx.doi.org/10.1016/j.jcp.2008.07.009.

[64] William B. Frakes and Kyo Kang. Software reuse research: Status and future. *IEEE Transactions on Software Engineering*, 31(7):529–536, 1995. http://dx.doi.org/10.1109/TSE.2005.85.

[65] Mark Galassi, Jim Davies, James Theiler, Brian Gough, Gerard Jungman, Patrick Alken, Michael Booth, and Fabrice Rossi. *GNU Scientific Library Reference Manual*. Network Theory Limited, Clifton, 3rd edition, 2009.

[66] Baskar Ganapathysubramanian and Nicholas Zabaras. Sparse grid collocation schemes for stochastic natural convection problems. *Journal of Computational Physics*, 225(1):652–685, 2007. http://dx.doi.org/10.1016/j.jcp.2006.12.014.

[67] Ronald Garcia, Jaakko Järvi, Andrew Lumsdaine, Jeremy Siek, and Jeremiah Willcock. An extended comparative study of language support for generic programming. *Journal of Functional Programming (JFP)*, 17(2):145–205, 2007. http://dx.doi.org/10.1017/S0956796806006198.

[68] David Garlan and Dewayne Perry. Introduction to the special issue on software architecture. *IEEE Transactions on Software Engineering*, 21(4):269–274, 1995. http://dl.acm.org/citation.cfm?id=205313.205314.

[69] Walter Gautschi. Construction of Gauss-Christoffel quadrature formulas. *Mathematics of Computation*, 22(102):251–270, 1968.

[70] Thomas Gerstner and Michael Griebel. Numerical integration using sparse grids. *Numerical Algorithms*, 18(3–4):209–232, 1998. http://dx.doi.org/10.1023/A:1019129717644.

[71] Thomas Gerstner and Michael Griebel. Dimension-adaptive tensor-product quadrature. *Computing*, 71(1):65–87, 2003. http://dx.doi.org/10.1007/s00607-003-0015-5.

[72] Roger G. Ghanem. The nonlinear Gaussian spectrum of log-normal stochastic processes and variables. *Journal of Applied Mechanics*, 66(4):964–973, 1999. http://dx.doi.org/10.1115/1.2791806.

[73] Roger G. Ghanem and Pol D. Spanos. *Stochastic Finite Elements: A Spectral Approach*. Dover Publications, Mineola, revised edition edition, 2003.

[74] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, Baltimore, 3rd edition, 1996.

[75] Peter Gottschling, David S. Wise, and Adwait Joshi. Generic support of algorithmic and structural recursion for scientific computing. *International Journal of Parallel, Emergent and Distributed Systems*, 24(6):479–503, 2009. http://dx.doi.org/10.1080/17445760902758560.

[76] Lori L. Graham, Kurtis Gurley, and Forrest Masters. Non-gaussian simulation of local material properties based on a moving-window technique. *Probabilistic Engineering Mechanics*, 18(3):223–234, 2003. http://dx.doi.org/10.1016/S0266-8920(03)00026-2.

[77] Lawrence L. Green, Hong-Zong Lin, and Mohammad R. Khalessi. Probabilistic methods for uncertainty propagation applied to aircraft design. In *20th AIAA Applied Aerodynamics Conference*, AIAA 2002–3140, pages 1–18, St. Louis, 2002. American Institute of Aeronautics and Astronautics. http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20030003828_2002153195.pdf.

[78] Douglas Gregor, Jaakko Järvi, Jeremy Siek, Bjarne Stroustrup, Gabriel Dos Reis, and Andrew Lumsdaine. Concepts: Linguistic support for generic programming in C++. In *ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA)*, volume 41, pages 291–310, Portland, 2006. Association for Comput-

ing Machinery (ACM). http://dx.doi.org/10.1145/1167515.1167499.

[79] Brian D. Hahn. *Essential MATLAB for Scientists and Engineers*. Butterworth-Heinemann, Oxford, 2nd edition, 2003.

[80] John. H. Halton. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik*, 2(1):84–90, 1960. http://dx.doi.org/10.1007/BF01386213.

[81] Lars U. Hansen, Wolfgang Heinze, and Peter Horst. Blended wing body structures in multidisciplinary pre-design. *Structural and Multidisciplinary Optimization*, 36(1):93–106, 2008. http://dx.doi.org/10.1007/s00158-007-0161-z.

[82] Wolfgang Heinze, Claudia M. Österheld, and Peter Horst. Multidisziplinäres Flugzeugentwurfsverfahren PrADO — Programmentwurf und Anwendung im Rahmen von Flugzeug-Konzeptstudien. In *Jahrbuch der DGLR-Jahrestagung 2001 in Hamburg*, Bonn, 2001.

[83] Rolf Henke, Tim Lammering, and Eckhard Anton. Impact of an innovative quiet regional aircraft on the air transportation system. *Journal of aircraft*, 47(3):875–886, 2010. http://dx.doi.org/10.2514/1.45785.

[84] Michi Henning and Steve Vinoski. *Advanced CORBA Programming with C++*. Addison-Wesley Professional Computing Series. Addison-Wesley, Boston, 1999.

[85] Nicholas J. Higham. *Functions of Matrices: Theory and Computation*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, 2008.

[86] Henry Hsieh. *Application of the PSUADE tool for sensitivity analysis of an engineering simulation*. Structural Mechanics Group, Lawrence Livermore National Laboratory, Livermore, 2007. https://computation.llnl.gov/casc/uncertainty_quantification/.

[87] Chao Hu and Byeng D. Youn. Adaptive-sparse polynomial chaos expansion for reliability analysis and design of complex engineering systems. *Structural and Multidisciplinary Optimization*, 43(3):419–442, 2011. http://dx.doi.org/10.1007/s00158-010-0568-9.

[88] David G. Hull. *Fundamentals of Airplane Flight Mechanics*. Springer, Berlin, 2007.

[89] Florian Hürlimann, Roland Kelm, Michael Dugas, Kim Oltmann, and Gerald Kress. Mass estimation of transport aircraft wingbox structures with a CAD/CAE-based multidisciplinary process. *Aerospace Science and Technology*, 15(4):323–333, 2011. http://dx.doi.org/10.1016/j.ast.2010.08.005.

[90] Kristina K. Jameson, David D. Marshall, Robert Ehrmann, Eric Paciano, Rory Golden, and Dave Mason. Design and wind tunnel testing of Cal Poly's AMELIA 10 foot span hybrid wing-body low noise CESTOL aircraft. In *27th International Congress of the Aeronautical Sciences*, pages 1–13, Nice, 2010. http://digitalcommons.calpoly.edu/aero_fac/58.

[91] Lloyd R. Jenkinson and James F. Marchman. *Aircraft Design Projects for Engineering Students*. AIAA Education Series. Butterworth Heinemann, Oxford, 2003.

[92] Chris Johnson, Steve Parker, David Weinstein, and Sean Heffernan. Component-based, problem-solving environments for large-scale scientific computing. *Concurrency and Computation: Practice and Experience*, 14(13–15):1337–1349, 2003. http://dx.doi.org/10.1002/cpe.693.

[93] Dominik Jürgens. Survey on software engineering for scientific applications — reuseable software, grid computing and application. Informatikbericht 2009–02, Institut für Wissenschaftliches Rechnen, Technische Universität Braunschweig, Braunschweig, 2009. http://www.digibib.tu-bs.de/?docid=00027815.

[94] Dominik Jürgens, Martin Krosche, and Rainer Niekamp. A process for stochastic material analysis based on empirical data. In *Technische Mechanik, Proceedings of the 2nd International Conference on Material Modelling*, volume 32 of *Scientific Journal for Fundamentals and Applications of Engineering Mechanics*, pages 303–306, Paris, 2012. http://www.uni-magdeburg.de/ifme/zeitschrift_tm/2012_Heft2_5/19_Juergens.pdf.

[95] David K. Kahaner and Giovanni Monegato. Nonexistence of extended Gauss-Laguerre and Gauss-Hermite quadrature rules with positive weights. *Jour-*

nal of Applied Mathematics and Physics (ZAMP), 29(6):983–986, 1978. http://dx.doi.org/10.1007/BF01590820.

[96] Christophe Kassiotis and Martin Hautefeuille. *coFeap's manual*. École Normale Supérieure de Cachan, Laboratoire de Mécanique et Technologies, Secteur Génie Civil, 2008.

[97] Arvinder Kaur and Kulvinder Singh Mann. Component based software engineering. *International Journal of Computer Applications*, 2(1):105–108, 2010. http://dx.doi.org/10.5120/605-855.

[98] Andreas Keese. *Numerical Solution of Systems with Stochastic Uncertainties - A General Purpose Framework for Stochastic Finite Elements*. PhD thesis, Technische Universität Braunschweig, Braunschweig, 2005. http://www.digibib.tu-bs.de/?docid=00001595.

[99] Andreas Keese and Hermann G. Matthies. Numerical methods and Smolyak quadrature for nonlinear stochastic partial differential equations. Informatikbericht 2003–5, Institut für Wissenschaftliches Rechnen, Technische Universität Braunschweig, Braunschweig, 2003. http://www.digibib.tu-bs.de/?docid=00001595.

[100] David Keith-Lucas. Defeating the square-cube law. *Flight International*, 94(3106):440–442, 1968.

[101] Alfred C. Kermode, Richard H. Barnard, and David R. Philpott. *Mechanics of Flight*. Pearson Prentice Hall, Upper Saddle River, 10th edition, 1996.

[102] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. Prentice Hall, Upper Saddle River, 2nd edition, 1988.

[103] Boris N. Khoromskij and Christoph Schwab. Tensor-structured Galerkin approximation of parametric and stochastic elliptic PDEs. *SIAM Journal on Scientific Computing*, 33(1):364–385, 2011. http://dx.doi.org/10.1137/100785715.

[104] Michelle R. Kirby and Dimitri N. Mavris. Forecasting technology uncertainty in preliminary aircraft design. In *4th World Aviation Congress and Exposition*, AIAA 99–01–5631, pages 1–12, San Francisco, 1999. Georgia Institute of Technology. http://hdl.handle.net/1853/6325.

[105] Carsten Koeppen and Udo Carl. Erfassung und Bewertung von Systemen im Flugzeugentwurf. In *Deutscher Luft- und Raumfahrtkongress (DGLR)*, Stuttgart, Germany, 2002. http://doku.b.tu-harburg.de/volltexte/2006/288/.

[106] Heiko Koziolek. Performance evaluation of component-based software systems: A survey. *Performance Evaluation*, 67(8):634–658, 2010. http://dx.doi.org/10.1016/j.peva.2009.07.007.

[107] Martin Krosche. Statistische Auswertung Numerischer Simulationen. Master's thesis, Institut für Wissenschaftliches Rechnen, Technische Universität Braunschweig, 2005. http://www.wire.tu-bs.de/mitarbeiter/martin_krosche/public/Diplomarbeit_Martin_Krosche.pdf.

[108] Martin Krosche and Martin Hautefeuille. Simulation and solution of stochastic systems with a component-based software design. In *Proceedings in Applied Mathematics and Mechanics (PAMM)*, volume 7, pages 2140001–2140002, Zurich, 2007. http://dx.doi.org/10.1002/pamm.200700067.

[109] Martin Krosche and Hermann G. Matthies. Component-based software system for simulating stochastic models with the Monte Carlo method. In *Multi-scale Computational Methods for Solids and Fluids*, ECCOMAS Thematic Conference, pages 136–141, Cachan, 2007. http://www.msf.ens-cachan.fr/pdf/MSFCachanEccomas2007.pdf.

[110] Martin Krosche and Hermann G. Matthies. Component-based software realisations of Monte Carlo and stochastic Galerkin methods. In *Proceedings in Applied Mathematics and Mechanics (PAMM)*, volume 8, pages 10765–10766, Bremen, 2008. http://dx.doi.org/10.1002/pamm.200810765.

[111] Martin Krosche and Rainer Niekamp. Low rank approximation in spectral stochastic finite element method with solution space adaption. Informatikbericht 2010–03, Institut für Wissenschaftliches Rechnen, Technische Universität Braunschweig, Braunschweig, 2010. http://www.digibib.tu-bs.de/?docid=00036351.

[112] Erasmus Langer. *Programmieren in Fortran*. Springer, Wien, 1993.

[113] Hans Petter Langtangen. *Python Scripting for Computational Science*. Texts

in Computational Science and Engineering. Springer, Berlin, 3rd edition, 2008.

[114] Kung-Kiu Lau and Zheng Wang. Software component models. *IEEE Transactions on Software Engineering*, 33(10):709–724, 2007. http://dx.doi.org/10.1109/TSE.2007.70726.

[115] Dirk P. Laurie. Anti-Gaussian quadrature formulas. *Mathematics of Computation*, 65(214):739–747, 1996. http://dx.doi.org/10.1090/S0025-5718-96-00713-2.

[116] Dirk P. Laurie. Computation of Gauss-type quadrature formulas. *Journal of Computational and Applied Mathematics*, 127(1–2):201–217, 2001. http://dx.doi.org/10.1016/S0377-0427(00)00506-9.

[117] Charles L. Lawson, Richard J. Hanson, David R. Kincaid, and Fred T. Krogh. Basic linear algebra subprograms for Fortran usage. *ACM Transactions on Mathematical Software (TOMS)*, 5(3):308–323, 1979. http://dx.doi.org/10.1145/355841.355847.

[118] Sophia Lefantzi, Jaideep Ray, and Habib N. Najm. Using the Common Component Architecture to design high performance scientific simulation codes. In *Parallel and Distributed Processing Symposium*, Washington, 2003. IEEE Computer Society. http://dx.doi.org/10.1109/IPDPS.2003.1213142.

[119] Richard B. Lehoucq, Danny C. Sorensen, and Chao Yang. *ARPACK Users' Guide: Solution of Large Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, 1998.

[120] Guang Lin, David Engel, and Paul Eslinger. Survey and evaluate uncertainty quantification methodologies. Technical report, Pacific Northwest National Laboratory, Richland, 2012.

[121] Alexander Litvinenko and Hermann G. Matthies. Sparse data formats and efficient numerical methods for uncertainties quantification in numerical aerodynamics. Informatikbericht 2010–01, Institut für Wissenschaftliches Rechnen, Technische Universität Braunschweig, Braunschweig, 2010. http://www.digibib.tu-bs.de/?docid=00036490.

[122] Alexander Litvinenko and Hermann G. Matthies. Uncertainties quantification and data compression in numerical aerodynamics. In *Proceedings in Applied Mathematics and Mechanics (PAMM)*, volume 11, pages 877–878, Weinheim, 2011. Wiley-VCH Verlag. http://dx.doi.org/10.1002/pamm.201110425.

[123] Dishi Liu. *Uncertainty Quantification with Shallow Water Equations*. PhD thesis, Technische Universität Braunschweig, Braunschweig, 2009. http://www.digibib.tu-bs.de/?docid=00032048.

[124] Jun S. Liu. *Monte Carlo Strategies in Scientific Computing*. Springer, New York, 2001.

[125] Michel Loève. *Probability Theory I & II*. Springer, Berlin, 1977/1978.

[126] Neal Madras. *Lectures on Monte Carlo Methods*. Fields Institute Monographs. American Mathematical Society, Providence, 2002.

[127] Olivier P. Le Maître, Habib N. Najm, Roger G. Ghanem, and Omar M. Knio. Multi-resolution analysis of Wiener-type uncertainty propagation schemes. *Journal of Computational Physics*, 197(2):502–531, 2004. http://dx.doi.org/10.1016/j.jcp.2003.12.020.

[128] Hermann G. Matthies. Stochastic finite elements: Computational approaches to stochastic partial differential equations. *ZAMM — Journal of Applied Mathematics and Mechanics*, 88(11):849–873, 2008. http://dx.doi.org/10.1002/zamm.200800095.

[129] Hermann G. Matthies and Andreas Keese. Galerkin methods for linear and nonlinear elliptic stochastic partial differential equations. *Computer Methods in Applied Mechanics and Engineering*, 194(12–16):1295–1331, 2005. http://dx.doi.org/10.1016/j.cma.2004.05.027.

[130] Hermann G. Matthies and Elmar Zander. Solving stochastic systems with low-rank tensor compression. *Linear Algebra and its Applications*, 436(10):3819–3838, 2012. http://dx.doi.org/10.1016/j.laa.2011.04.017.

[131] Dimitri N. Mavris and Daniel A. DeLaurentis. A probabilistic approach for examining aircraft concept feasibility and viability. *Aircraft Design*, 3(2):79–101, 2000. http://dx.doi.org/10.1016/S1369-8869(00)00008-2.

[132] Dimitri N. Mavris, Daniel A. DeLaurentis, Oliver Bandte, and Mark A. Hale. A stochastic approach to multi-disciplinary aircraft analysis and design. In *36th Aerospace Sciences Meeting & Exhibit*, AIAA 98–0912, pages 1–15, Reno, 1998. Georgia Institute of Technology. http://hdl.handle.net/1853/6387.

[133] Dimitri N. Mavris, Daniel A. DeLaurentis, and Danielle Suzanne Soban. Probabilistic assessment of handling qualities constraints in aircraft preliminary design. In *36th Aerospace Sciences Meeting & Exhibit*, AIAA 98–0492, pages 1–13, Reno, 1998. Georgia Institute of Technology. http://hdl.handle.net/1853/6276.

[134] Dimitri N. Mavris, Noel I. Macsotai, and Bryce A. Roth. A probabilistic design methodology for commercial aircraft engine cycle selection. In *3rd World Aviation Congress and Exposition*, AIAA 98–5510, pages 1–9, Anaheim, 1998. Georgia Institute of Technology. http://hdl.handle.net/1853/6377.

[135] M. Douglas McIlroy. Mass produced software components. *Scientific Affairs Division, NATO, Brussels, Germany*, pages 138–155, 1969.

[136] Michael D. McKay, Richard J. Beckman, and William J. Conover. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics*, 21(2):239–245, 1979.

[137] Karl Meerbergen, Krešimir Fresl, and Toon Knapen. C++ bindings to external software libraries with examples from BLAS, LAPACK, UMFPACK, and MUMPS. *ACM Transactions on Mathematical Software (TOMS)*, 36(4):1–23, 2009. http://dx.doi.org/10.1145/1555386.1555391.

[138] Loujaine Mehrez, Alireza Doostan, David Moens, and Dirk Vandepitte. Stochastic identification of composite material properties from limited experimental databases, part II: Uncertainty modelling. *Mechanical Systems and Signal Processing*, 27:484–498, 2012. http://dx.doi.org/10.1016/j.ymssp.2011.09.001.

[139] Loujaine Mehrez, David Moens, and Dirk Vandepitte. Stochastic identification of composite material properties from limited experimental databases, part I: Experimental database construction. *Mechanical Systems and Signal Processing*, 27:471–483, 2012. http://dx.doi.org/10.1016/j.ymssp.2011.09.004.

[140] K. Jarrod Millman and Michael Aivazis. Python for scientists and engineers. *Computing in Science and Engineering*, 13(2):9–12, 2011. http://dx.doi.org/10.1109/MCSE.2011.36.

[141] Giovanni Monegato. An overview of the computational aspects of Kronrod quadrature rules. *Numerical Algorithms*, 26(2):173–196, 2001. http://dx.doi.org/10.1023/A:1016640617732.

[142] Douglas C. Montgomery. *Design and Analysis of Experiments*. John Wiley & Sons, Hoboken, 6th edition, 2005.

[143] Tarek El Moselhy and Youssef Marzouk. An adaptive iterative method for high dimensional stochastic PDEs. Technical report, Aerospace Computational Design Laboratory, Aeronautics & Astronautics Department, Massachusetts Institute of Technology, Cambridge, under preparation.

[144] David R. Musser and Alexander A. Stepanov. Generic programming. In *International Symposium on Symbolic and Algebraic (ISSAC)*, pages 13–25, Rome, 1988.

[145] Habib N. Najm. Uncertainty quantification and polynomial chaos techniques in computational fluid dynamics. *Annual Review of Fluid Mechanics*, 41:35–52, 2009. http://dx.doi.org/10.1146/annurev.fluid.010908.165248.

[146] Habib N. Najm. Uncertainty quantification in fluid flow. In *Turbulent Combustion Modeling*, volume 95 of *Fluid Mechanics and Its Applications*, pages 381–407. Springer, Berlin, 2011. http://dx.doi.org/10.1007/978-94-007-0412-1_16.

[147] Daniel Neufeld, Joon Chung, and Kamran Behdinian. Aircraft conceptual design optimization considering fidelity uncertainties. *Journal of Aircraft*, 48(5):1602–1612, 2011. http://dx.doi.org/10.2514/1.C031312.

[148] M. F. Ngah and A. Young. Application of the spectral stochastic finite element method for performance prediction of composite structures. *Composite Structures*, 78(3):447–456, 2007. http://dx.doi.org/10.1016/j.compstruct.2005.11.009.

[149] Harald Niederreiter. Low-discrepancy and low-dispersion sequences. *Jour-*

nal of Number Theory, 30(1):51–70, 1988. http://dx.doi.org/10.1016/0022-314X(88)90025-X.

[150] Harald Niederreiter. *Random Number Generation and Quasi-Monte Carlo Methods*. CBMS-NSF Regional Conference Series in Applied Mathematics. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, 1992.

[151] Rainer Niekamp. CTL manual for Linux/Unix for the usage with C++. Technical report, Institut für Wissenschaftliches Rechnen, Technische Universität Braunschweig, Braunschweig. http://www.wire.tu-bs.de/forschung/projekte/ctl/e_ctl.html.

[152] Rainer Niekamp. *Verteilte Algorithmen für h-, p- und d-adaptive Berechnungen in der nichtlinearen Strukturmechanik*. PhD thesis, Institut für Baumechanik und Numerische Mechanik, Universität Hannover, Hannover, 2000.

[153] Fabio Nobile, Raúl Tempone, and Clayton G. Webster. An anisotropic sparse grid stochastic collocation method for partial differential equations with random input data. *SIAM Journal on Numerical Analysis*, 46(5):2411–2442, 2008. http://dx.doi.org/10.1137/070680540.

[154] Fabio Nobile, Raúl Tempone, and Clayton G. Webster. A sparse grid stochastic collocation method for partial differential equations with random input data. *SIAM Journal on Numerical Analysis*, 46(5):2309–2345, 2008. http://dx.doi.org/10.1137/060663660.

[155] Hyuk-Chun Noh and Taehyo Park. Response variability of laminate composite plates due to spatially random material parameter. *Computer Methods in Applied Mechanics and Engineering*, 200(29–32):2397–2406, 2011. http://dx.doi.org/10.1016/j.cma.2011.03.020.

[156] Anthony Nouy. A generalized spectral decomposition technique to solve a class of linear stochastic partial differential equations. *Computer Methods in Applied Mechanics and Engineering*, 196(45–48):4521–4537, 2007. http://dx.doi.org/10.1016/j.cma.2007.05.016.

[157] Anthony Nouy. Generalized spectral decomposition method for solving stochastic finite element equations: Invariant subspace problem and dedicated algorithms. *Computer Methods in Applied Mechanics and Engineer-*

ing, 197(51–52):4718–4736, 2008. http://dx.doi.org/10.1016/j.cma.2008.06.012.

[158] Anthony Nouy. Recent developments in spectral stochastic methods for the numerical solution of stochastic partial differential equations. *Archives of Computational Methods in Engineering*, 16(3):251–285, 2009. http://dx.doi.org/10.1007/s11831-009-9034-5.

[159] Anthony Nouy. Proper generalized decompositions and separated representations for the numerical solution of high dimensional stochastic problems. *Archives of Computational Methods in Engineering*, 17(4):403–434, 2010. http://dx.doi.org/10.1007/s11831-010-9054-1.

[160] Cosmin E. Oancea, Jason W. A. Selby, Mark W. Giesbrecht, and Stephen M. Watt. Distributed models of thread-level speculation. In *International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, PDPTA '05, pages 920–927, Las Vegas, 2005.

[161] Cosmin E. Oancea and Stephen M. Watt. Parametric polymorphism for software component architectures. In *ACM Object Oriented Programming, Systems, Languages and Applications (OOPSLA)*, volume 40 of *OOPSLA '05*, pages 147–166, San Diego, 2005. Association for Computing Machinery (ACM). http://dx.doi.org/10.1145/1103845.1094823.

[162] Cosmin E. Oancea and Stephen M. Watt. An architecture for generic extensions. *Science of Computer Programming*, 76(4):258–277, 2011. http://dx.doi.org/10.1016/j.scico.2009.09.008.

[163] Bernd Oestereich. *Objektorientierte Softwareentwicklung: Analyse und Design mit der Unified Modeling Language*. Oldenbourg Verlag, München, 5th edition, 2001.

[164] James Douglas Orton and Karl E. Weick. Loosely coupled systems: A reconceptualization. *The Academy of Management Review*, 15(2):203–223, 1990.

[165] Edoardo Patelli, Gerhart I. Schuëller, Helmut J. Pradlwarter, Marcos A. Valdebenito, H. Murat Panayirci, Barbara Goller, Matteo Broggi, and Pierre Beaurepaire. COSSAN-X: A general purpose code for computational stochastic structural analysis. In *IV European Conference on Computational Mechanics*, Paris, 2010.

[166] Thomas N. L. Patterson. The optimum addition of points to quadrature formulae. *Mathematics of Computation*, 22(104):847–856, 1968. http://dx.doi.org/10.1090/S0025-5718-68-99866-9.

[167] Thomas N. L. Patterson. Algorithm 468: Algorithm for automatic numerical integration over a finite interval [d1]. *Communications of the ACM*, 16(11):694–699, 1973. http://dx.doi.org/10.1145/355611.362543.

[168] Fernando Pérez, Brian E. Granger, and John D. Hunter. Python: An ecosystem for scientific computing. *Computing in Science and Engineering*, 13(2):13–21, 2011. http://dx.doi.org/10.1109/MCSE.2010.119.

[169] Knut Petras. On the Smolyak cubature error for analytic functions. *Advances in Computational Mathematics*, 12(1):71–93, 2000. http://dx.doi.org/10.1023/A:1018904816230.

[170] Knut Petras. Asymptotically minimal Smolyak cubature. Technical report, Institut für Angewandte Mathematik, Technische Universität Braunschweig, Braunschweig, 2001.

[171] Eileen P. Poeter, Mary C. Hill, Edward R. Banta, Steffen Mehl, and Steen Christensen. *UCODE_2005 and six other computer codes for universal sensitivity analysis, calibration, and uncertainty evaluation*, 2008. http://igwmc.mines.edu/freeware/ucode/.

[172] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C++: The Art of Scientific Computing*. Cambridge University Press, Cambridge, 2nd edition, 2002.

[173] Mark Price, Srinivasan Raghunathan, and Richard Curran. An integrated systems engineering approach to aircraft design. *Progress in Aerospace Sciences*, 42(4):331–376, 2006. http://dx.doi.org/10.1016/j.paerosci.2006.11.002.

[174] Alfio Quarteroni and Fausto Saleri. *Scientific Computing with MATLAB and Octave*. Texts in Computational Science and Engineering. Springer, Berlin, 2nd edition, 2006.

[175] Rolf Radespiel, Kai-Christoph Pfingsten, and Christoph Jensch. Flow analysis of augmented high-lift systems. *Notes on Numerical Fluid Mechan-*

ics and Multidisciplinary Design, 102:168–189, 2009. http://dx.doi.org/10.1007/978-3-540-95998-4_10.

[176] Thiagarajan Ravichandran and Marcus A. Rothenberger. Software reuse strategies and component markets. *Communications of the ACM*, 46(8):109–114, 2003. http://dx.doi.org/10.1145/859670.859678.

[177] Daniel P. Raymer. *Aircraft Design: A Conceptual Approach*. AIAA education series, Washington, 4th edition, 2006.

[178] Daniel Reckzeh. Aerodynamic design of the high-lift-wing for a megaliner aircraft. *Aerospace Science and Technology*, 7(2):107–119, 2003. http://dx.doi.org/10.1016/S1270-9638(02)00002-0.

[179] Darren Redfern and Edgar Chandler. *Maple O.D.E. Lab Book*. Springer, New York, 1996.

[180] Gabriel Dos Reis and Jaakko Järvi. What is generic programming? In *Proceedings of the First International Workshop on Library-Centric Software Design (LCSD '05)*, San Diego, 2005. ACM Object Oriented Programming, Systems, Languages and Applications (OOPSLA), Association for Computing Machinery (ACM).

[181] Ken F. Riley, Michael P. Hobson, and Stephen J. Bence. *Mathematical Methods for Physics and Engineering: A Comprehensive Guide*. Cambridge University Press, Cambridge, 1997.

[182] Murray Rosenblatt. Remarks on a multivariate transformation. *The Annals of Mathematical Statistics*, 23(3):470–472, 1952. http://dx.doi.org/10.1214/aoms/1177729394.

[183] Eveline Rosseel, Timotheus Boonen, and Stefan Vandewalle. Algebraic multigrid for stationary and time-dependent partial differential equations with stochastic coefficients. *Numerical Linear Algebra with Applications*, 15(2–3):141–163, 2008. http://dx.doi.org/10.1002/nla.568.

[184] Yousef Saad. *Iterative Methods for Sparse Linear Systems*. PWS Publishing Company, Boston, 1st edition, 1996.

[185] Shigehiro Sakamotoa and Roger Ghanem. Simulation of multi-dimensional

non-Gaussian non-stationary random fields. *Probabilistic Engineering Mechanics*, 17(2):167–176, 2002.

[186] Conrad Sanderson. Armadillo: An open source C++ linear algebra library for fast prototyping and computationally intensive experiments. Technical report, NICTA, St Lucia, 2010.

[187] Douglas C. Schmidt. Why software reuse has failed and how to make it work for you. `http://www.cs.wustl.edu/~schmidt/reuse-lessons.html`, 1999. edited 2006.

[188] Oliver Schneider, Stefan Kreth, and Lothar Bertsch. Towards a quiet short take-off and landing transportation system: Concept evaluation and ATM integration. In *International Powered Lift Conference*, Philadelphia, 2010.

[189] Gerhart I. Schuëller and Helmut J. Pradlwarter. Computational stochastic structural analysis (COSSAN) — a software tool. *Structural Safety*, 28(1–2):68–82, 2006. `http://dx.doi.org/10.1016/j.strusafe.2005.03.005`.

[190] Kolja Seeckt, Wolfgang Heinze, and Dieter Scholz. Hydrogen powered freighter aircraft — the final results of the green freighter project. In *Proceedings of the International Congress of the Aeronautical Sciences (ICAS)*, Niza, 2010.

[191] Irving E. Segal and Ray A. Kunze. *Integrals and Operators*. A Series of Comprehensive Studies in Mathematics. Springer, Berlin, 2nd edition, 1978.

[192] C. Siva, M. Senthil Murugan, and Ranjan Ganguli. Uncertainty quantification in helicopter performance using Monte Carlo simulations. *Journal of Aircraft*, 48(5):1503–1511, 2011. `http://dx.doi.org/10.2514/1.C000288`.

[193] Christian Soize. Non-Gaussian positive-definite matrix-valued random fields for elliptic stochastic partial differential operators. *Computer Methods in Applied Mechanics and Engineering*, 195(1–3):26–64, 2006. `http://dx.doi.org/10.1016/j.cma.2004.12.014`.

[194] Alexander Stepanov and Meng Lee. The Standard Template Library. Technical report, Hewlett-Packard Laboratories, Palo Alto, 1995.

[195] Bjarne Stroustrup. *The C++ Programming Language*. Addison-Wesley, Reading, 3rd edition, 1997.

[196] Bjarne Stroustrup. C and C++: Case studies in compatibility. *The C/C++ Users Journal*, pages 1–6, 2002.

[197] Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, London, 2nd edition, 2002.

[198] Brian D. Taylor. Umbral presentations for polynomial sequences. *Computers & Mathematics with Applications*, 41(9):1085–1098, 2001. `http://dx.doi.org/10.1016/S0898-1221(01)00083-9`.

[199] Robert L. Taylor. FEAP - a finite element analysis program. `www.ce.berkeley.edu/feap`.

[200] Vladimir N. Temlyakov. Greedy approximation. In *Acta Numerica*, volume 17, pages 235–409. Cambridge University Press, Cambridge, 2008. `http://dx.doi.org/10.1017/S0962492906380014`.

[201] Radu A. Todor and Christoph Schwab. Convergence rates for sparse chaos approximations of elliptic problems with stochastic coefficients. *IMA Journal of Numerical Analysis*, 27(2):232–261, 2007. `http://dx.doi.org/10.1093/imanum/drl025`.

[202] Lloyd N. Trefethen. Is Gauss quadrature better than Clenshaw-Curtis? *SIAM Review*, 50(1):67–87, 2008. `http://dx.doi.org/10.1137/060659831`.

[203] Ulrich Trottenberg, Cornelis W. Oosterlee, and Anton Schüller. *Multigrid*. Academic Press, London, 2001.

[204] Aslak Tveito, Hans Petter Langtangen, Bjørn Frederik Nielsen, and Xing Cai. *Elements of Scientific Computing*, chapter 6. Texts in Computational Science and Engineering. Springer, Berlin, 2010.

[205] Elisabeth Ullmann. A Kronecker product preconditioner for stochastic Galerkin finite element discretizations. *SIAM Journal on Scientific Computing*, 32(2):923–946, 2010. `http://dx.doi.org/10.1137/080742853`.

[206] Erik Vanmarcke. *Random Fields: Analysis and Synthesis*. The MIT Press, Cambridge, 1988.

[207] Todd L. Veldhuizen. Blitz++: The library that thinks it is a compiler. In *Advances in Software Tools for Scientific Computing*, volume 10 of *Lecture Notes in Computational Science and Engineering*, pages 57–87, Berlin, 2000. Springer. http://dx.doi.org/10.1007/978-3-642-57172-5_2.

[208] Xiaoliang Wan and George E. Karniadakis. An adaptive multi-element generalized polynomial chaos method for stochastic differential equations. *Journal of Computational Physics*, 209(2):617–642, 2005. http://dx.doi.org/10.1016/j.jcp.2005.03.023.

[209] Terrence A. Weisshaar. The design of a high capacity long range cargo aircraft — final report. NASA contractor report NASA–CR–197176, Purdue University, School of Aeronautics and Astronautics, West Lafayette, 1994.

[210] Piotr Wendykier and James G. Nagy. Parallel Colt: A high-performance Java library for scientific computing and image processing. *ACM Transactions on Mathematical Software (TOMS)*, 37(3):1–22, 2010. http://dx.doi.org/10.1145/1824801.1824809.

[211] Christian Werner-Westphal, Wolfgang Heinze, and Peter Horst. Structural sizing for an unconventional, environment-friendly aircraft configuration within integrated conceptual design. *Aerospace Science and Technology*, 12(2):184–194, 2008. http://dx.doi.org/10.1016/j.ast.2007.05.006.

[212] Norbert Wiener. The homogeneous chaos. *American Journal of Mathematics*, 60(4):897–936, 1938. http://dx.doi.org/10.2307/2371268.

[213] Stephen Wolfram. *The Mathematica Book*. Wolfram Media/Cambridge Univerity Press, Champaign, 3rd edition, 1996.

[214] Ann Wollrath, Roger Riggs, and Jim Waldo. A distributed object model for the Java system. In *Proceedings of the USENIX, Second USENIX Conference on Object-Oriented Technologies (COOTS)*, pages 219–232, Toronto, 1996.

[215] David C. Wynn, Khadidja Grebici, and P. John Clarkson. Modelling the evolution of uncertainty levels during design. *International Journal on Interactive*

Design and Manufacturing, 5(3):187–202, 2011. http://dx.doi.org/10.1007/s12008-011-0131-y.

[216] Dongbin Xiu and Jan S. Hesthaven. High-order collocation methods for differential equations with random inputs. *SIAM Journal on Scientific Computing*, 27(3):1118–1139, 2005. http://dx.doi.org/10.1137/040615201.

[217] Dongbin Xiu and George E. Karniadakis. The Wiener-Askey polynomial chaos for stochastic differential equations. *SIAM Journal on Scientific Computing*, 24(2):619–644, 2002. http://dx.doi.org/10.1137/S1064827501387826.

[218] Dongbin Xiu and George E. Karniadakis. Modeling uncertainty in flow simulations via generalized polynomial chaos. *Journal of Computational Physics*, 187(1):137–167, 2003. http://dx.doi.org/10.1016/S0021-9991(03)00092-5.

[219] Wen Yao, Xiaoqian Chen, Wencai Luo, Michel van Tooren, and Jian Guo. Review of uncertainty-based multidisciplinary design optimization methods for aerospace vehicles. *Progress in Aerospace Sciences*, 47(6):450–479, 2011. http://dx.doi.org/10.1016/j.paerosci.2011.05.001.

[220] Jian Yin, Khushbu Agarwal, Manoj Krishnan, Daniel Chavarría-Miranda, Ian Gorton, and Tom Epperly. Implementing high performance remote method invocation in CCA. *IEEE International Conference on Cluster Computing*, pages 547–551, 2011. http://dx.doi.org/10.1109/CLUSTER.2011.78.

[221] Thomas A. Zang, Michael J. Hemsch, Mark W. Hilburger, Sean P. Kenny, James M. Luckring, Peiman Maghami, Sharon L. Padula, and W. Jefferson Stroud. Needs and opportunities for uncertainty-based multidisciplinary design methods for aerospace vehicles. Technical Report TM–2002–211462, NASA, Langley Research Center, Hampton, 2002. http://aircraftdesign.nuaa.edu.cn/MDO/ref/Uncertainty/General%20Topics/NASA-2002-tm211462.pdf.

List of Publications Related to this Thesis

The author of this thesis is the main or joint author of a number of publications which are related to this thesis.

Concepts of the distributed generic component-based software architecture in Chapter 3 to simulate probabilistic models are addressed in several publications. The corresponding framework for a direct integration is introduced in [1, 2] by means of the MC method. A concrete deterministic simulator component in [1] interfaces a third-party simulation code, which is written in MATLAB. The technical background to do so is published in [8]. The SGM-based framework is presented in [4], in addition to the one for the MC method. Analogously to the deterministic simulator component in the mentioned architecture, the concept of a deterministic simulator component for fluid dynamics in the context of fluid-structure interactions is proposed in [3, 5].

The VLR-SR1U, VLR-OPT, and RBSSE schemes in Section 2.7.2 to enable a low-rank approximation of the solution and an adaptive construction of the solution space are presented in [9]. The composite material in Section 4.2 is recently introduced in [6, 7].

Proceedings:

[1] Martin Krosche and Martin Hautefeuille. Simulation and solution of stochastic systems with a component-based software design. In *Proceedings in Applied Mathematics and Mechanics (PAMM)*, volume 7, pages 2140001–2140002, Zurich, 2007. http://dx.doi.org/10.1002/pamm.200700067.

[2] Martin Krosche and Hermann G. Matthies. Component-based software system for simulating stochastic models with the Monte Carlo method. In *Multi-scale Computational Methods for Solids and Fluids*, ECCOMAS Thematic Conference, pages 136–141, Cachan, 2007. http://www.msf.ens-cachan.fr/pdf/MSFCachanEccomas2007.pdf.

[3] Joachim Rang, Martin Krosche, Rainer Niekamp, and Hermann G. Matthies. Solving fluid-structure interaction problems using MpCCI and the component template library (CTL). In *Proceedings of the 8th MpCCI User Forum*, pages 36–41, Sankt Augustin, 2007. http://www.mpcci.de/fileadmin/mpcci/Userforum/MpCCI_8th_UserForum.pdf.

[4] Martin Krosche and Hermann G. Matthies. Component-based software realisations of Monte Carlo and stochastic Galerkin methods. In *Proceedings in Applied Mathematics and Mechanics (PAMM)*, volume 8, pages 10765–10766, Bremen, 2008. http://dx.doi.org/10.1002/pamm.200810765.

[5] Joachim Rang, Martin Krosche, Rainer Niekamp, and Hermann G. Matthies. Solving fluid-structure interaction problems using strong coupling algorithms with the component template library (CTL). In *Proceedings in Applied Mathematics and Mechanics (PAMM)*, volume 8, pages 10987–10988, Bremen, 2008. http://dx.doi.org/10.1002/pamm.200810987.

[6] Dominik Jürgens, Rainer Niekamp, and Martin Krosche. A multi-scale framework for stochastic analysis of multi-phase material. In *1st International Symposium on Uncertainty Modelling in Engineering (ISUME)*, pages 71–72, Prague, 2011. http://klobouk.fsv.cvut.cz/~anicka/isume/ISUME2011_proceedings.pdf.

[7] Dominik Jürgens, Martin Krosche, and Rainer Niekamp. A process for stochastic material analysis based on empirical data. In *Technische Mechanik, Proceedings of the 2nd International Conference on Material Modelling*, volume 32 of *Scientific Journal for Fundamentals and Applications of Engineering Mechanics*, pages 303–306, Paris, 2012. http://www.uni-magdeburg.de/ifme/zeitschrift_tm/2012_Heft2_5/19_Juergens.pdf.

Technical Reports:

[8] Boris Bügling and Martin Krosche. Coupling the CTL and MATLAB. Informatikbericht 2007–03, Institut für Wissenschaftliches Rechnen, Technische Universität Braunschweig, Braunschweig, 2007. http://www.digibib.tu-bs.de/?docid=00021292.

[9] Martin Krosche and Rainer Niekamp. Low rank approximation in spectral stochastic finite element method with solution space adaption. Informatikbericht 2010–03, Institut für Wissenschaftliches Rechnen, Technische Universität Braunschweig, Braunschweig, 2010. http://www.digibib.tu-bs.de/?docid=00036351.